视频直播

播放器 SDK

文档版本	01
发布日期	2020-11-25

华为技术有限公司

商标声明


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

注意

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

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

目 录

1 SDK 介绍

华为云视频直播播放器和点播播放器是合并的，提供了Web端播放器和移动端播放器，移动端播放器包含iOS版和Android版。播放器SDK除了提供点播和直播的基本播放功能外，还具备书签播放、倍速播放、加密播放等常用功能。

功能介绍

播放器SDK支持直播和点播播放，具体支持的功能如**表1-1**所示。

表 1-1 功能列表

功能类别	功能	描述
播放器	播放器中常用功能。	• 支持直播和点播。 • 支持调整显示比例：默认、原始大小、16:9、4:3、铺满屏幕、居中裁剪。 • 支持滑动调节播放进度、声音、亮度；双击播放、暂停；保存播放进度。 • 支持边播边缓存。 • 支持倍速播放。 • 支持视频截图。 • 支持镜像显示。 • 支持锁屏功能，包含锁定旋转和隐藏界面元素。 • 支持断网自动重连。 • 支持弹幕。 • 支持多种封装格式（MP4、m3u8、flv视频格式和mp3音频格式）。 • 多种视频编码格式（h264、h265、mpeg4、mjpeg）。 • 支持多种流媒体协议（RTSP、RTMP、HLS、concat协议等）。 • 支持播放本地视频以及raw和assets视频。 • 支持重力感应自动进入、退出全屏以及手动进入、退出全屏，全屏状态下可锁定。 • 完美实现列表播放（RecyclerView和ListView），列表自动播放。 • 支持列表小窗全局悬浮播放，Android 8.0画中画功能。 • 支持连续播放一个列表的视频。 • 支持广告播放。 • 支持清晰度切换。 • 支持完全自定义控制层。 • 支持多路播放器同时播放，没有任何控制UI的纯播放无缝衔接播放Demo。 • 支持内置和外挂字幕。

应用场景

- 场景一：短视频列表滑动及循环播放
  - 场景描述

    在短视频应用中，往往采用全屏滑动播放的方式展示精彩内容，华为云视频播放器SDK提供视图大小自定义功能可以简单实现全屏播放的需求，同时提

供多实例、自动播放和预加载能力，可轻松实现多视频全屏滑动播放功能。由于视频内容短而精，短视频应用通常采用循环播放的方式让用户反复观看，为了节约用户流量和无缝的循环播放，华为云视频播放器SDK提供了边播边缓存和循环播放接口，只需简单设置即可满足应用场景。

- 使用流程

  使用华为云**短视频SDK**录制视频，然后使用**上传SDK**上传至点播服务，最后使用华为云视频播放器SDK的demo框架，开启边播边缓存和循环播放功能实现短视频列表的全屏滑动播放和循环播放功能。

- 场景二：视频版权保护
  - 场景描述

    现在对于视频版权的保护意识和要求越来越高，例如用户要做一个教育类的视频网站，由教师提供视频课程，只有购买课程的用户才能观看，那么如何保护视频不被盗播和盗版？华为云视频播放器SDK提供多层级保护：一、提供防盗链功能仅允许配置了白名单的用户访问；二、提供URL鉴权功能，保护视频仅能在鉴权有效期内播放器；三、提供加密流播放功能，保障视频仅能使用华为云视频播放器SDK才能播放；四、提供安全下载功能，保证下载的视频仅能通过在控制台配置的唯一应用（bundleID或签名）播放。

  - 使用流程

    首先在点播控制台安全设置中开启防盗链和URL鉴权功能，然后在**转码设置**中配置的转码流中包含加密流，再在下载设置中开启安全下载并生成加密文件，最后把加密文件集成到播放器SDK，使用播放器SDK的demo框架进行播放和下载。

Demo 体验

您可以通过扫描二维码的方式，安装华为视频云APP Demo，它提供了直播推流、播放器、短视频、上传视频等功能 ，具体请参见**Demo体验**。

华为视频云服务同时提供了Web播放器Demo，可实现点播和直播播放，**体验地址**。

2 SDK 下载

移动端（Android版和iOS版）播放器SDK已集成到华为视频云APP Demo中，您可以在**SDK开发者中心**下载视频云APP Demo源码进行快速集成。Web端播放器SDK请参考**开发指南**引入对应js文件，集成开发即可。

- 视频云APP Android版

- 视频云APP iOS版

3 Web 播放器

3.1 概述

华为云Web播放器hwplayer是基于Video.js框架来进行拓展和开发的，满足播放器主流功能，如字幕、倍速播放等，简洁轻量，易于开发使用。同时支持Html5和Flash播放，支持自定义皮肤和扩展插件。

须知

- Web播放器还未完全适配移动端浏览器，暂无法保障在移动端浏览器中的功能，且暂不支持播放H.265编码的视频，若播放可能导致黑屏等问题。
- 若需要兼容IE8，请进行兼容性配置。

```
<!--[if lt IE 9]><script src="https://media-cache.huaweicloud.com/video/hwplayer/1.4.0/lib/video-js-5.20.5/ie8/videojs-ie8.min.js"></script><![endif]-->
```

支持的视频格式

表 3-1 支持的视频格式

视频格式	直播	点播	示例链接	PC端	移动端
MP4	×	√	https://xxx,vod.huaweicloud.com/asset/xxx.mp4	√	√
FLV	√	√	https://xxx,vod.huaweicloud.com/asset/xxx.flv	√	×
HLS	√	√	https://xxx,vod.huaweicloud.com/asset/xxx.m3u8	√	√
RTMP	√	×	rtmp://xxx.hwcloudlive.com/live/xxx	IE 8+：√ 其他浏览器：×	×

3.2 初始化

您可以通过以下步骤在页面中创建一个播放器。

Step1：引入 js 文件

- 在页面中引入Web播放器脚本文件。
  ```
  <script src="https://media-cache.huaweicloud.com/video/hwplayer/1.4.0/dist/hwplayer.js"></script>
  ```
- 若播放器需要支持FLV格式，请在引入“hwplayer.js”前，先引入“flv-1.4.2.min.js”文件，并在“hwplayer.js”引入中增加“flvjs=true”属性。

  须知

  若在同一个页面实现js上传和Web播放，需要注意flv.js共存导致的视频无法播放等问题。

  ```
  <script src="https://media-cache.huaweicloud.com/video/hwplayer/1.4.0/lib/flv-1.4.2.min.js"></script>
  <script src="https://media-cache.huaweicloud.com/video/hwplayer/1.4.0/dist/hwplayer.js?flvjs=true"></script>
  ```

  调用src方法更新视频源时需要加上type参数。

  ```
  hwPlayer.src({
      src: url,//视频or直播url
      type: 'video/flv'
  })
  ```
- 若播放器需要支持DASH格式，请在引入“hwplayer.js”前，先引入“dash-2.9.2.all.min.js”文件，并在“hwplayer.js”引入中增加“dashjs=true”属性。

```
<script src="https://media-cache.huaweicloud.com/video/hwplayer/1.4.0/lib/dash-2.9.2.all.min.js"></
script>
<script src="https://media-cache.huaweicloud.com/video/hwplayer/1.4.0/dist/hwplayer.js?
dashjs=true"></script>
```

Step2：初始化播放器

播放器支持使用js方式和标签方式进行初始化，建议您优先选择js方式。

- js方式（推荐方式）

```
<video id="test" class="video-js vjs-default-skin vjs-big-play-centered"></video>
<script>
    hwplayerloaded(function () {
        var options = {
            //是否显示控制栏，包括进度条，播放暂停按钮，音量调节等组件
            controls: true,
            width: 640,
            height: 360,
        };
        var player = new HWPlayer('test', options, function () {
            //播放器已经准备好了
            player.src("https://35.cdn-vod.huaweicloud.com/asset/ba4f5df688f4ed6f569470d688ec4a22/
c5d8003cb1d108035d3a902adb2bc5cc.mp4");
            // "this"指向的是HWPlayer的实例对象player
            player.play();
            // 使用事件监听
            player.on('ended', function () {
            //播放结束了
            });
        });
    });
</script>
```

this.src：请填写为在**视频点播控制台**或**视频直播控制台**中获取的点播视频播放地址或直播播放地址。

须知

调用src方法时，除mp4格式的视频外，都需要标注视频类型。

例如：视频播放url是一个m3u8的视频，则调用接口方法为player.src({src:url,type:"application/x-mpegURL"})。具体详情请参考**播放器属性和接口**中的src方法。

视频格式和对应的type：

- m3u8：application/x-mpegURL
- mp4：video/mp4
- flv：video/flv
- mpd：application/dash+xml
- rtmp：rtmp/flv

HWPlayer构造函数接收如下参数：

- **id**：video标签dom的id。
- **options**：播放器的配置选项，具体配置项说明请参见**表3-2**。
- **onready**：播放器初始化完成后的回调函数，可使用this引用播放器的实例对象，如this.play(),this.pause()，this.on('ended')等操作。

- 标签方式

在<video>标签中加上class="video-js"和data-setup='{}'属性。其中，**source src**请填写为在**视频点播控制台**或**视频直播控制台**中获取的点播视频播放地址或直播播放地址。

```
<video id="test" class="video-js vjs-default-skin" controls preload="none" width="640" height="264" poster="http://vjs.zencdn.net/v/oceans.png" data-setup='{}'>
    <source src="https://35.cdn-vod.huaweicloud.com/asset/ba4f5df688f4ed6f569470d688ec4a22/c5d8003cb1d108035d3a902adb2bc5cc.mp4" type="video/mp4">
</video>
<script>
     hwplayerloaded(function(){});
</script>
```

- **data-setup**：必选，若为空值，请填写data-setup='{}'。
- **hwplayerloaded**：必选，加载播放器在**hwplayerloaded**方法触发时执行。

代码示例

- js方式（MP4格式）

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>demo</title>
</head>
<body>
<video id="test" class="video-js vjs-default-skin vjs-big-play-centered">
</video>
<!--[if lt IE 9]><script src="https://media-cache.huaweicloud.com/video/hwplayer/1.4.0/lib/video-js-5.20.5/ie8/videojs-ie8.min.js"></script><![endif]--><SCRIPT src="//media-cache.huaweicloud.com/video/hwplayer/1.4.0/dist/hwplayer.js"></SCRIPT>
<script>
    hwplayerloaded(function () {
        var player = new HWPlayer('test', {
            controls: true,
            userId: 'playerDemo01',
            domainId: 'hwPlayer',
            width: 640,
            height: 360,
            playbackRates: [.5, 1, 1.5, 2]
        });
        player.src("https://35.cdn-vod.huaweicloud.com/asset/ba4f5df688f4ed6f569470d688ec4a22/c5d8003cb1d108035d3a902adb2bc5cc.mp4");
    });
</script>
</body>
</html>
```

- js方式（FLV格式）

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>demo</title>
</head>
<body>
<video id="test" class="video-js vjs-default-skin vjs-big-play-centered">
</video>
<script src="https://media-cache.huaweicloud.com/video/hwplayer/1.4.0/lib/flv-1.4.2.min.js"></script>
<script src="https://media-cache.huaweicloud.com/video/hwplayer/1.4.0/dist/hwplayer.js?flvjs=true"></script>
<script>
    hwplayerloaded(function () {
        var player = new HWPlayer('test', {
            controls: true,
            userId: 'playerDemo01',
            domainId: 'hwPlayer',
            width: 640,
```

```
            height: 360,
            playbackRates: [.5, 1, 1.5, 2]
        });
         player.src({
            src: url, //视频地址
            type: 'video/flv'
         });
    });
</script>
</body>
</html>
```

- 标签方式

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>demo</title>
</head>
<body>
<video controls id="test" class="video-js vjs-default-skin vjs-big-play-centered" data-setup='{"width":
640,"height":360,"playbackRates":[0.5, 1, 1.5, 2]}'>
    <source src="https://35.cdn-vod.huaweicloud.com/asset/ba4f5df688f4ed6f569470d688ec4a22/
c5d8003cb1d108035d3a902adb2bc5cc.mp4" type="video/mp4">
</video>
<!--[if lt IE 9]><script src="//media-cache.huaweicloud.com/video/hwplayer/1.4.0/lib/video-
js-5.20.5/ie8/videojs-ie8.min.js"></script><![endif]--><SCRIPT src="//media-cache.huaweicloud.com/
video/hwplayer/1.4.0/dist/hwplayer.js"></SCRIPT>
<script>
    hwplayerloaded(function () {});
</script>
</body>
</html>
```

说明

由于浏览器的跨域安全限制，以上代码无法在本地HTML文件上调试，请将需要调试的文件上传到服务器端再进行访问。

3.3 常用功能说明

华为云提供的Web播放器除了提供点播和直播的基本播放功能外，还支持点播书签播放、倍速播放、清晰度切换等。直播播放暂不支持这些功能。

书签播放

1. 在退出播放前获取和保存当前播放位置，也可以设置定时间隔保存。可将书签保存至本地缓存或上传至服务器（云端书签）。

```
videoPlayer.currentTime(); //获取当前播放位置
```

2. 再次播放时，需要先从本地缓存或服务器上获取播放地址，并设置当前播放时间，即可从上次播放位置开始播放了。

```
videoPlayer.currentTime(time); //time为你此前缓存的播放位置
```

倍速播放

华为Web播放器支持倍速播放功能，暂仅支持IE8e及以上版本的IE浏览器，支持的播放格式暂只有MP4。

- 可在播放器初始化时配置倍速菜单，推荐设置倍速值范围是0.5-2。

```
var videoPlayer = new HWPlayer(id, {
    playbackRates: [1, 2, 3]
});
```

- 若有需要，也可以自行调用倍速方法。

```
videoPlayer.playbackRate();  //获取当前播放倍速
videoPlayer.playbackRate（2）;//双倍速度播放当前视频
```

清晰度切换

播放器提供清晰度切换功能，具体使用方法如下所示：

```
//设置视频清晰度切换
videoPlayer.updateSrc([
    {
        src: "//path/to/OP.mp4",
        type: 'video/mp4',
        label: '原画'
    },
    {
        src: "//path/to/HD.mp4",
        type: 'video/mp4',
        label: '高清'
    },
    {
        src: "//path/to/SD.mp4",
        type: 'video/mp4',
        label: '标清'
    },
    {
        src: "//path/to/FLUENT.mp4",
        type: 'video/mp4',
        label: '流畅'
    }
]);
```

实现后的效果如下所示：

加载字幕

播放器支持加载在线字幕到当前视频中。

```
player.loadSubtitles(url);//加载在线字幕，加载后会在播放器控制栏显示字幕菜单按钮
```

url为字幕文件的地址，必须为webvtt格式的字幕。

字幕支持调整字体等样式：

```
player.textTrackSettings.setValues({
    backgroundColor: "#F00",//背景色
   backgroundOpacity: "1",//背景透明度
   color: "#000",//字体颜色
   fontFamily: "proportionalSansSerif",//字体
   textOpacity: "1",//字体透明度
   windowColor: "#000",//
    windowOpacity: "0"//
})
player.textTrackDisplay.updateDisplay();//设置完字体样式后，必须刷新字幕
```

其中，相关参数的取值如下：

```
const color = [
    "#000", //黑色
   "#FFF", //白色
   "#F00", //红色
   "#0F0", //绿色
   "#00F", //蓝色
   "#FF0", //黄色
   "#F0F", //紫红色
   "#0FF", //青色
]
const opacity = [
    0.5,//半透明
   1,//不透明
   0//透明
]
```

3.4 播放器属性和接口

属性

可以通过以下两种方式设置相关属性：

- 通过video标签的data-setup属性。
  ```
  <video data-setup='{"autoplay":"true",.....}'
  ```
- 通过构造函数HWPlayer中的optiongs设置（推荐方式）。
  ```
  var player = new HWPlayer('test',{autoplay:true,....})
  ```

播放器常用属性如表3-2所示。更多options参数配置说明请参见Video.js。

表 3-2 常用属性列表

参数	参数说明
autoplay	播放器准备好之后，是否自动播放。 取值为true/false，默认为false。
controls	是否有控制条。 取值为true/false。
height	表示视频容器的高度。 取值为字符串或数字，单位为像素。 例如：height:300或者height:'300px'。
width	表示视频容器的宽度。 取值为字符串或数字，单位为像素。

参数	参数说明
loop	视频播放结束后，是否循环播放。 取值为true/false。
muted	是否静音。 取值为true/false。
poster	播放前显示的视频画面，播放开始后自动移除。通常传入一个URL。
playbackRates	倍速播放设置。 取值为数组类型，播放器已默认开启倍速播放显示开关并配置常用倍速，若有特殊需求也可自行设置，比如直播不需倍速可设置为[]。 例如：[0.5, 1, 1.5, 2]
preload	预加载。 取值为字符串，如下所示： ● auto（缺省值）：自动。 ● metadata：元数据信息 ，比如视频长度，尺寸等。 ● none：不预加载任何数据，直到用户开始播放才开始下载。

sources	表示资源文件，取值类型Array。
	例如：
	`new HWPlayer('test',{`
	`  sources: [{`
	`  src: '//path/to/video.mp4',`
	`  type: 'video/mp4'`
	`  }, {`
	`  src: '//path/to/video.webm',`
	`  type: 'video/webm'`
	`  }]`
	`});`

waiting	在一个待执行的操作（如回放）因等待另一个操作（如跳跃或下载）被延迟时触发。例如：缓冲视频。

方法

播放器初始化后，常使用的方法如表3-3所示。具体API说明请参考Player Methods。

表 3-3 常用方法列表

方法	参数	说明
play()	无	启动播放。
pause()	无	暂停播放。
currentTime()	无	获取当前播放时间。

方法	参数	说明
currentTime(time)	time：跳转时间，单位为秒。	设置当前播放时间，表示跳转到某个时间进行播放。
duration()	无	获取播放视频的时长。
src()	无	获取播放视频的地址。
src({src:url,type:type})	• url：视频地址，如vodtest.mp4。 • type：视频类型，必须设置，否则可能无法解析视频。	设置播放视频的地址。 视频格式和对应的type： • m3u8：application/x-mpegURL • mp4：video/mp4 • flv：video/flv • mpd：application/dash+xml • rtmp：rtmp/flv
poster()	无	获取播放视频的封面地址。
poster(picture)	picture：图片地址，该地址需要设置允许跨域访问。	设置播放视频的封面地址。

updateSrc(sharpness)
说明
直播播放暂不支持该方法。
	sharpness：视频清晰度，包含如下参数：
	• src：视频地址。
	• type：视频格式类型，如“video/mp4”。
	• label：显示的清晰度标签，可设置为“原画”、“高清”、“标清”、“流畅”。
		设置视频清晰度切换。
		videoPlayer.updateSrc([
		{
		src: "//path/to/OP.mp4",
		type: 'video/mp4',
		label: '原画'
		},
		{
		src: "//path/to/HD.mp4",
		type: 'video/mp4',
		label: '高清'
		},
		{
		src: "//path/to/SD.mp4",
		type: 'video/mp4',
		label: '标清'
		},
		{
		src: "//path/to/FLUENT.mp4",
		type: 'video/mp4',
		label: '流畅'
		}
		]);

loadSubtitles(url)	url：字幕文件的地址。	加载字幕。

事件

播放器支持的常用事件如表3-4所示，事件的相关详情说明请参见Video.js。

表 3-4 常用事件列表

事件	说明
loadstart	在媒体开始加载时触发。
loadedmetadata	媒体的元数据已经加载完毕，现在所有的属性包含了它们应有的有效信息。
loadeddata	媒体的第一帧已经加载完毕。
play	播放。
firstplay	首次播放。
progress	当浏览器开始下载音视频时触发。当前已下载的总计信息可以在元素的buffered属性中获取。
timeupdate	播放时间更新。
pause	暂停。
ended	播放结束。
error	播放报错。
resize	视频尺寸改变。
fullscreenchange	切换全屏。
durationchange	视频时长改变。
volumechange	音量变化。

- 通过播放器实例的on方法绑定事件。
  ```
  videoPlayer.on(“play”, function(){
  });
  ```
- 通过播放器实例的removeEvent移除事件。
  ```
  videoPlayer.removeEvent(“eventName”, myFunc);
  ```

3.5 常用样式说明

暂停时显示播放按钮

```
.vjs-paused.vjs-big-play-button,
.vjs-paused.vjs-has-started.vjs-big-play-button {
display: block;
}
```

单击屏幕播放/暂停

```
.video-js.vjs-playing.vjs-tech {
pointer-events: auto;
}
```

进度显示当前播放时间

默认倒序显示时间，也就是视频播放的剩余时间。若需要显示当前的播放时间及视频总时长，可通过以下样式解决：

```
.video-js.vjs-time-control{display:block;}
.video-js.vjs-remaining-time{display: none;}
```

重载视频文件

```
var video = document.getElementById('example_video');
$("#reload").click(function() {
    video.pause()
    video.setAttribute('src', 'b.mp4');
    video.load();
    video.play();
});
```

3.6 自定义皮肤

播放器的皮肤由HTML和CSS构建，自定义皮肤简单方便，具体请参见**videojs皮肤设计**。

3.7 跨域配置

目前点播服务和新版直播服务默认启用跨域，无需再进行配置。若您使用的域名仍部署在旧版直播服务上，则需要**提交工单**修改跨域配置，您可以参见**使用前必读**查询域名是否部署在旧版视频直播服务上。

HTML5 跨域播放视频

使用CDN成功推流后，如果视频在浏览器上无法播放且控制台含以下报错信息：

```
blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
```

请检查视频的跨域配置，需要在HTTP头添加如下字段：

```
access-control-allow-methods: GET, HEAD, PUT, POST, DELETE, OPTIONS
access-control-allow-origin: *
```

Flash 跨域播放视频

如果播放时出现如**图3-1**所示界面，请先确认该视频地址和视频类型是否正确，若确认正确，且在Android和iOS的播放器上都可以正常播放，则可能是跨域问题导致。需要做如下处理：

添加crossdomain.xml文件到视频地址域名的根目录下，xml文件内容如下：

```
<cross-domain-policy>
<allow-access-from domain="*" secure="false"/>
<allow-http-request-headers-from domain="*" headers="*"/>
</cross-domain-policy>
```

图 3-1 视频播放提示失败界面

封面图片跨域

如果封面图片加载失败，可能是跨域问题导致，需要配置封面图片服务器允许跨域，具体配置请参见**HTML5跨域播放视频**。

4 Android 播放器

4.1 开发前准备

软硬件环境配置要求

播放器SDK需要集成到APP工程中，单独的SDK对软硬件环境没有特别的要求。

- 准备Android Studio或者Eclipse集成开发环境。
- 准备Android运行环境，Android 5.0以上的设备。
- 支持的终端CPU架构：armV7、x86。

说明

当前播放器在Sony电视上可能会存在自适应切换码率播放时，出现一帧黑屏的问题。 验证型号：电视索尼SONY KD-55X7000D 55英寸高清4K HDR安卓6.0系统。

SDK 集成

本文档以Android Studio方式为例。

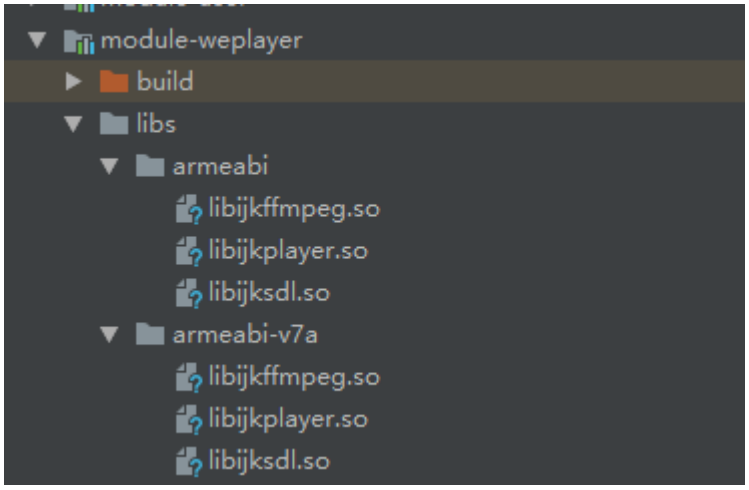
说明

如果开启混淆，需要在当前工程的混淆文件中针对工程引用的第三方jar包做忽略混淆，忽略jar包中对应包名。

步骤1 解压**下载**的SDK压缩包，获取播放器模块“module-weplayer”。

步骤2 在“Android_版本号_日期 > module-weplayer”目录下按不同架构（armv7/arm64-v8a ）将对应二进制so库文件拷贝到APP工程的libs下对应目录中，如**图4-1**所示。

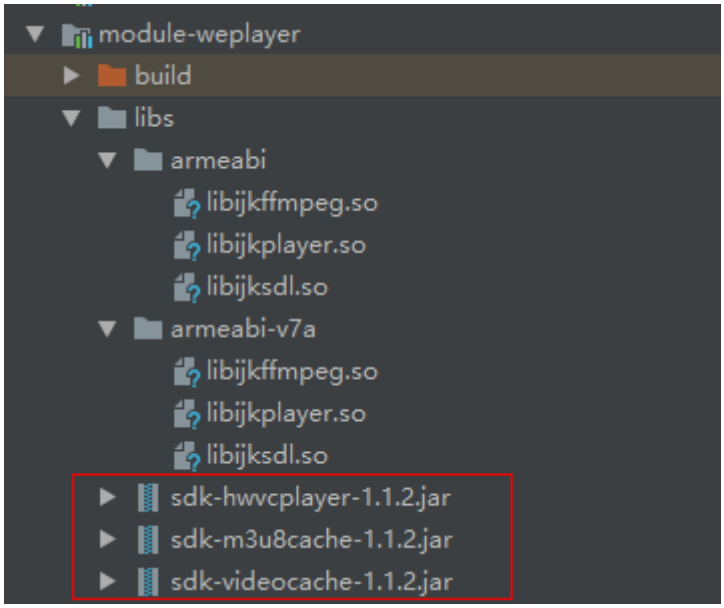
图 4-1 APP 工程目录



步骤3 将“module-weplayer\libs”目录下的如下jar包添加到APP工程的libs文件夹下，如图4-2所示。

- sdk-hwvcplayer-1.1.2.jar
- sdk-m3u8cache-1.1.2.jar
- sdk-videocache-1.1.2.jar

图 4-2 库文件



步骤4 修改APP的build.gradle文件，添加配置和依赖。

```
android {
  ......
  defaultConfig {
    ......
    ndk {
      abiFilters "armeabi-v7a" // 指定要ndk需要兼容的架构(这样其他依赖包里mips,x86,armeabi,arm-v8之类的so会被过滤掉)}
    }
  }
  ......
  sourceSets {
```

```
    main {
      jniLibs.srcDirs = ['libs']
    }
  }
  ......
}

dependencies {
implementation files('libs/sdk-hwvcplayer-1.1.2.jar')
implementation files('libs/sdk-m3u8cache-1.1.2.jar')
implementation files('libs/sdk-videocache-1.1.2.jar')
implementation "com.alibaba:fastjson:1.2.62"
}
```

步骤5 选择“Sync Project With Gradle Files”。

至此，播放器SDK已集成到工程中。

----结束

配置 APP 权限

在AndroidManifest.xml文件的manifest标签中申明网络、外部存储访问、摄像头、闪光灯、录音等权限。

```
<!—权限说明-->

<!-- 允许程序打开网络套接字 -->
<uses-permission android:name="android.permission.INTERNET"/>
<!-- 允许程序使用PowerManager WakeLocks以防止处理器休眠或者屏幕锁屏 -->
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<!-- 允许程序向外部存储设备写数据 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<!-- 允许程序向外部存储设备读数据 -->
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<!-- 允许程序获取网络相关信息 -->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<!-- 允许程序获取Wifi网络状态信息 -->
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
```

4.2 SDK 使用

实现播放功能

步骤1 创建播放器。

```
mMediaPlayer = new IjkMediaPlayer(Context context,String domainId,String userId,Boolean isLive);
mMediaPlayer.native_setLogLevel(IjkMediaPlayer.IJK_LOG_DEBUG);
mMediaPlayer.setOnPreparedListener(onPreparedListener);
```

- **context**：上下文。
- **isLive**：是否为直播。
- **domainId**：租户id，自定义，不支持中文字符，用于播放器数据统计。
- **userId**：用户id，自定义，不支持中文字符，用于播放器数据统计。

步骤2 设置播放地址。

```
mMediaPlayer.setDataSource(mAppContext, url, headers);
```

- **mAppContext**：上下文。
- **url**：播放片源的地址，为快速起播，该参数可以设为""字符串。

- **headers**：请求头信息，没有可设置为null。

说明

直播服务必须设置片源URL，在播放HLS时移片源时，必须将直播地址与时移地址用“|”进行拼接，格式为“直播地址”|“时移地址”。

步骤3 设置显示承载的VIEW。

播放器支持两种view（SurfaceView和TextureView），SurfaceView采用setDisplay接口，TextureView采用setSurface接口。

- SurfaceView采用setDisplay接口
```
mMediaPlayer.setDisplay(SurfaceHolder holder);
```
- TextureView采用setSurface接口
```
mMediaPlayer.setSurface(Surface surface);
```

说明

播放加密视频尽量使用surfaceView。非加密视频可以使用TextureView。surfaceview是新建了一个新的window，不受原始的view的一切属性约束，只有View是可用的，才能设置给播放器，否则无法播放。

步骤4 准备播放。

使用prepareAsync函数。

```
mMediaPlayer.prepareAsync();
```

准备完成后会触发OnPreparedListener监听事件。

步骤5 结束播放。

```
if (mMediaPlayer != null)
mMediaPlayer.release();
```

----结束

拖拽播放

播放GOP过大的视频时，调用seekTo会跳回到拖动前的位置，这是因为视频的关键帧问题（GOP导致的），视频压缩率比较高，而seek只支持关键帧，出现这个情况就是原始的视频文件中i帧比较少，播放器会在拖动的位置找最近的关键帧。

```
 public void seekTo(long time) {
try {
mMediaPlayer.seekTo((int) time);
} catch (IllegalStateException e) {
      e.printStackTrace();
}
}
```

外挂字幕功能

步骤1 加载字幕文件。

```
private SubtitleReader mSubtitleReader;
private List<SubtitleLineInfo> mSubtitleLineInfos;
private SubtitleView mSubtitleView;
//加载文件
InputStream inputStream = getResources().openRawResource(R.raw.shame_ass);
AssSubtitleFileReader assSubtitleFileReader = new AssSubtitleFileReader();
SubtitleInfo subtitleInfo = assSubtitleFileReader.readInputStream(inputStream);
mSubtitleReader = new SubtitleReader();
```

```
mSubtitleReader.setSubtitleInfo(subtitleInfo);
mSubtitleLineInfos = subtitleInfo.getSubtitleLineInfos();
```

步骤2 根据进度条的改变显示对应的字幕。

```
long progress = getCurrentPosition();
int lineNumber = SubtitleUtil.getLineNumber(mSubtitleLineInfos, progress,
mSubtitleReader.getPlayOffset());
    if (lineNumber == -1) {
        mSubtitleView.setText("");
    }
    else {
        SubtitleLineInfo subtitleLineInfo = mSubtitleLineInfos.get(lineNumber);
        mSubtitleView.setText(Html.fromHtml(subtitleLineInfo.getSubtitleHtml()))
    };
```

----结束

弹幕功能

步骤1 添加依赖。

```
dependencies {
...
implementation 'com.github.ctiao:DanmakuFlameMaster:0.9.25'
}
```

步骤2 新建播放器并添加配置。

```
mDefinitionWeVideoView.openDanmukuView(true);
```

步骤3 设置弹幕可见与隐藏。

```
public void setDanmuku(boolean isShow) {
        if(mDanmakuView == null){
            initDanMuView();
         }else if(isShow){
            mDanmakuView.show();
        }else {
            mDanmakuView.hide();
        }
}
```

步骤4 发表弹幕。

```
mDefinitionWeVideoView.addDanmaku("普通字幕");
mDefinitionWeVideoView.addDanmakuWithDrawable("自定义字幕"，drawable);
```

----结束

m3u8 边播边缓存

- 初始化

```
private EncryptM3U8Server m3u8Server = new EncryptM3U8Server();//加密版m3u8 httpServer
private String encryptKey = "63F06F99D823D33AAB89A0A93DECFEE0"; //get the key by
AES128Utils.getAESKey()
m3u8Server.execute();//启动服务

private void initM3u8Manger() {
dirPath = StorageUtils.getCacheDirectory(this).getPath() + "/m3u8Cache";//缓存目录
//common config !
M3U8CacheConfig.build(getApplicationContext())
.setSaveDir(dirPath)
.setDebugMode(true);
// add listener
M3U8Cache.getInstance().setOnM3U8CacheListener(onM3U8CacheListener);
M3U8Cache.getInstance().setEncryptKey(encryptKey);//encryptKey 为空则不会加密
}
```

- 生命周期


```
@Override
protected void onPause() {
super.onPause();
m3u8Server.encrypt();//加密m3u8
}

@Override
protected void onResume() {
super.onResume();
m3u8Server.decrypt();//解密m3u8
}

@Override
protected void onDestroy() {
super.onDestroy();
m3u8Server.finish();//关闭服务
}
```

- 边播边缓存功能使用

```
M3U8Cache.getInstance().cache(url);//开始缓存m3u8视频,再次调用则为暂停 调用cache2(url)则不会触发暂停只有开始缓存功能
url = M3U8Cache.getInstance().getM3U8Path(url);
m3u8 = m3u8Server.createLocalHttpUrl(url);//获取本地代理的m3u8地址 传递给播放器进行边播边缓存
```

播放器优化

- 软硬编解码的选择。

表 4-1 软硬编解码说明

名称	描述	优点	缺点
软编解码	主要使用CPU进行编解码，通常情况下使用FFmpeg进行编码和解压音视频数据。	• 在不同的设备、系统版本中兼容性极高。 • 解码时，色彩还原度更高。 • 编解码过程可扩展性强。	CPU占用高，手机易发热，耗电量大。
硬编解码	主要使用非CPU进行编解码，如GPU等，通常情况下直接调用系统API进行音视频编解码处理。	系统占用少，执行效率高。	• 兼容性低，需根据硬件厂商和系统版本单独适配。 • 可控性比较差。

  - iOS硬件芯片为统一的videoToolBox解码后的NV12数据格式。
  - Android使用的mediacodec解码的数据格式跟手机硬件厂商有关，无法统一mediacodec解码出来的数据格式。

  综合以上情况，在推流方面，iOS系统和硬件设备统一性高，使用全硬编方案效果更好；Android因机型繁杂，支持程度不一，推荐4.3以上版本使用硬编。在播放解码方面，iOS硬解和软解支持性都较高，软解功耗更高，但是在部分细节方面表现较优，可控性强，具体视项目情况选择；Android推荐4.1版本以上使用硬解，4.1以下版本使用软解。

- 直播与点播建议设置参数如下所示。

```
if (mIsLive) {
// Param for living
ijkMediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER, "max_cached_duration", 3000);//最大缓存大小是3秒，可以根据实际需求进行修改。
ijkMediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER, "infbuf", 1);//无限读。
ijkMediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER, "packet-buffering", 0);//关闭播放器缓冲。
} else {
// Param for playback
ijkMediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER, "max_cached_duration", 0);
ijkMediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER, "infbuf", 0);
ijkMediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER, "packet-buffering", 1);
}
```

- 累积延迟。
  - RTMP到达CDN时存在4.0到5.0秒的延迟，可以通过设置IJKPlayer相关参数进行优化。

    表 4-2 参数说明

参数	说明
skip_loop_filter	对所选帧进行跳过环路滤波。
skip_frame	编解码器丢弃所有非关键帧。

  - 优化延迟问题还可以从推流端将编码器调低GOP，如1秒一个GOP，这样可以减小延迟时长，但会导致编码器压缩率降低，图像质量下降，并且会加大服务端的数据压力。
  - 导致延迟还有一个主要原因是ffplay默认的帧率控制没有追帧策略，若网络发生抖动，延时会累加。优化策略就是调整帧率控制部分，保障流畅度和实时性。

监听函数实现

具体的监听器函数以及参数说明请参见**监听器接口**。

实现函数如下：

```
mMediaPlayer.setOnPreparedListener(onPreparedListener);
mMediaPlayer.setOnVideoSizeChangedListener(onSizeChangedListener);
mMediaPlayer.setOnCompletionListener(onCompletionListener);
mMediaPlayer.setOnErrorListener(onErrorListener );
mMediaPlayer.setOnInfoListener(onInfoListener);
mMediaPlayer.setOnBufferingUpdateListener(onBufferingUpdateListener);
mMediaPlayer.setOnSeekCompleteListener(onSeekCompleteListener);
mMediaPlayer.setOnTimedTextListener(OnTimedTextListener);
```

括号内为实现了播放器事件监听器的类：OnPreparedListener、OnCompletionListener、OnBufferingUpdateListener、OnErrorListener、OnInfoListener、OnSeekListener、OnVideoSizeChangedListener，您需要根据实际需求实现这些监听器类接口。

```
例如OnErrorListener:
private IMediaPlayer.OnErrorListener onErrorListener = new IMediaPlayer.OnErrorListener() {
@Override
public boolean onError(IMediaPlayer iMediaPlayer, int framework_err, int impl_err) {
//根据实际需求实现对应的功能。
return true;
}
};
```

播放器挂起与恢复

应用切换至后台时调用播放器pause挂起播放。

```
@Override
public void surfaceDestroyed(SurfaceHolder holder){
 if (player!= null)
 {
   mMediaPlayer.pause();
  }
}
```

应用切换至前台时调用播放器start继续播放。

```
@Override
public void surfaceCreated(SurfaceHolder holder)
{
 if (mMediaPlayer != null)
 {
   mMediaPlayer.start();
 }
}
```

错误码处理

您可以通过player.setOnErrorListener接口设置错误监听，当播放器出现错误导致无法继续播放时，会通过设置的监听回调函数onError获取详细的错误码。

调节音量和静音

设置播放器音量或者静音。

```
mMediaPlayer.setVolume(float leftVolume, float rightVolume)
```

音量设置范围：0.0f-1.0f，0.0f为静音。

切换片源

片源播放结束或者退出播放后，释放播放器。

```
if (mMediaPlayer != null)
mMediaPlayer.release();
```

然后执行一遍**实现播放功能**。

4.3 接口参考

IjkMediaPlayer 类

IjkMediaPlayer主要包含播放相关的方法。

方法	描述
public void setDataSource(String path, Map<String, String> headers); public void setDataSource(Context context, Uri uri) public void setDataSource(Context context, Uri uri, Map<String, String> headers) public void setDataSource(String path) public void setDataSource(FileDescriptor fd) private void setDataSource(FileDescriptor fd, long offset, long length) public void setDataSource(IMediaDataSource mediaDataSource)	【功能说明】 打开播放器写本地文件日志功能。 【请求参数】 ● path：文件的路径，或您要播放的流的http/rtsp URL。 ● headers：播放的流的http请求关联的标头。 ● context：解析URI时使用的Context。 ● uri：要播放的数据的内容URI。 ● fd：要播放的FileDescriptor文件。 ● mediaDataSource：要播放的IMediaDataSource文件。 【返回参数】 无 【注意事项】 无
public boolean isPlaying()	【功能说明】 判断当前是否处于播放状态。 【请求参数】 无 【返回参数】 boolean型数据：是否处于播放状态。 【注意事项】 无
public seekTo(long time)	【功能说明】 拖拽播放。 【请求参数】 time：拖拽位置。 【返回参数】 无 【注意事项】 无

方法	描述
public long getTcpSpeed()	【功能说明】 获取下载速度。 【请求参数】 time：拖拽位置。 【返回参数】 long型数据:下载速度。 【注意事项】 无
public void setSpeed(float speed)	【功能说明】 设置播放速度。 【请求参数】 speed：变速设置。 **取值范围**：0.5f-2.0f。 【返回参数】 无 【注意事项】 无
public float getSpeed()	【功能说明】 获取播放速度。 【请求参数】 无 【返回参数】 float型数据：获取变速参数。 【注意事项】 无
setOption	具体请参见**表4-3**。

表 4-3 setOption 参数说明

分类	参数	描述	示例
OPT_CATEGORY_PLAYER	an	是否输出音频。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER,"an",1)
OPT_CATEGORY_PLAYER	vn	是否输出视频。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER,"vn",1)
OPT_CATEGORY_PLAYER	nodisp	是否禁止图像显示，只输出音频。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER,"nodisp",1)

分类	参数	描述	示例
OPT_CATEGORY_PLAYER	flush_packets	每处理一个packet之后刷新io上下文。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER,"flush_packets",1)
OPT_CATEGORY_PLAYER	isModifyTone	设置是否开启变调。 ● 0：开启 ● 1：不开启	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER,"soundtouch",isModifyTone?0:1)
OPT_CATEGORY_CODEC	skip_loop_filter	设置是否开启环路过滤。 ● 0：开启，画面质量高，解码开销大。 ● 48：关闭，画面质量差点，解码开销小。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_CODEC,"skip_loop_filter",0)
OPT_CATEGORY_FORMAT	analyzemaxduration	设置播放前的最大探测时间。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_FORMAT,"analyzemaxduration",100L)
OPT_CATEGORY_FORMAT	analyzeduration	设置播放前的探测时间。 设置为1，达到首屏秒开效果。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_FORMAT,"analyzeduration",1)
OPT_CATEGORY_FORMAT	probesize	播放前的探测Size，默认是设置值小一点播出画面会更快。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_FORMAT,"probesize",1024*10)
OPT_CATEGORY_FORMAT	flush_packets	每处理一个packet之后刷新io上下文。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_FORMAT,"flush_packets",1L)
OPT_CATEGORY_PLAYER	packet-buffering	是否开启预缓冲，一般直播项目会开启，达到秒开的效果，但会导致播放丢帧卡顿。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER,"packet-buffering",isBufferCache?1:0)

分类	参数	描述	示例
OPT_CATEGORY_PLAYER	reconnect	播放重连次数。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER,"reconnect",5)
OPT_CATEGORY_PLAYER	max-buffer-size	最大缓冲大小。 单位：kb。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER,"max-buffer-size",maxCacheSize);
OPT_CATEGORY_PLAYER	framedrop	跳帧处理。 CPU处理视频较慢时，进行跳帧处理，保证画面和声音同步。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER,"framedrop",5)
OPT_CATEGORY_PLAYER	max-fps	最大fps。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER,"max-fps",30)
OPT_CATEGORY_PLAYER	mediacodec	设置硬解码方式。 jkPlayer支持硬解码和软解码。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER, "mediacodec", 1)
OPT_CATEGORY_PLAYER	mediacodec-auto-rotate	软解码时不会旋转视频角度，需要通过设置onInfo的what == IMediaPlayer.MEDIA_INFO_VIDEO_ROTATION_CHANGED获取角度，进行画面旋转。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER, "mediacodec-auto-rotate", 1)
OPT_CATEGORY_PLAYER	mediacodec-handle-resolution-change	是否自动处理分辨率更改。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER, "mediacodec-handle-resolution-change", 1)

分类	参数	描述	示例
OPT_CATEGORY_PLAYER	enable-accurate-seek	SeekTo设置优化。 因为原始的视频文件中i帧较少，而seek只支持关键帧，导致视频在SeekTo的时候，会跳回到拖动前的位置。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER, "enable-accurate-seek", 1)
OPT_CATEGORY_FORMAT	fflags	解决m3u8文件拖动问题。 如播放时长为3小时左右的音频文件，从开始播放几秒拖动到2小时左右时，要loading 10分钟。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_FORMAT, "fflags", "fastseek")
OPT_CATEGORY_PLAYER	enable-accurate-seek	设置之后，高码率m3u8的播放出现卡顿，声音画面不同步，或者只有画面，没有声音，或者声音画面不同步。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_PLAYER, "enable-accurate-seek", 1)
OPT_CATEGORY_FORMAT	probesize	播放前的探测Size，默认是1M，设置数值小于1M画面播放速度更快。	mediaPlayer.setOption(IjkMediaPlayer.OPT_CATEGORY_FORMAT, "probesize", 1024 * 10)

监听器接口

播放器的各状态都是通过客户端注册监听，播放器以响应监听的方式来上报的。客户端的监听响应函数需要继承播放器的监听器接口，并编写接口方法实现。

客户端需要在监听响应时，通过播放器的状态和上报的参数并根据APP设计的需要来决定后续如何处理。以下是对播放器各监听器接口的说明。

表 4-4 监听器函数以及参数说明

接口名	描述
public void onPrepared(IMediaPlayer iMediaPlayer)	播放器准备好播放。可以在该接口的回调函数调用播放器的start方法。
public void onVideoSizeChanged(IMediaPlayer iMediaPlayer, int i, int i1, int i2, int i3)	视频宽高值变化监听。客户端可以在该监听接口的回调函数获取片源的宽和高。
public void onCompletion(IMediaPlayer iMediaPlayer)	播放完成。客户端可以在该监听接口的回调函数进行播放完成的操作。
public boolean onError(IMediaPlayer iMediaPlayer, int framework_err, int impl_err)	错误上报，错误码信息，具体请参考**表4-5**。客户端可以在该监听接口的回调函数对播放器遇到的错误进行相应操作。
public boolean onInfo(IMediaPlayer iMediaPlayer, int what, int extra)	消息上报，具体请参见**表4-6**。客户端可以在该监听接口的回调函数对播放器上报的报警等其他信息进行相应处理。
public void onBufferingUpdate(IMediaPlayer iMediaPlayer, int percent)	缓冲更新。客户端可以在该监听接口的回调函数获取缓冲完成的百分比。
public void setOnSeekCompleteListener(IMediaPlayer iMediaPlayer)	拖拽完成监听。客户端可以在该监听接口的回调函数获取缓冲完成的百分比。
public void onTimedText(IMediaPlayer iMediaPlayer, IjkTimedText ijkTimedText)	字幕监听接口。

表 4-5 media_error_type 枚举

枚举成员	描述
MEDIA_ERROR_UNKOWN	1，未指明的错误类型。
int MEDIA_ERROR_SERVER_DIED	100，媒体服务关闭或异常退出，视频中断，通常情况是视频源异常或者不支持的视频类型。
MEDIA_ERROR_NOT_VALID_FOR_PROGRESSIVE_PLAYBACK	200，数据错误没有有效的回收。
MEDIA_ERROR_IO	-1004，IO错误。
MEDIA_ERROR_MALFORMED	-1007，比特流不符合编码标准或者文件规格。

枚举成员	描述
MEDIA_ERROR_UNSUPPORTED	-1010，比特流符合编码标准或者文件规格，但是媒体框架不支持。
MEDIA_ERROR_TIMED_OUT	-110，数据超时Error (-10000,0)。
MEDIA_INFO_VIDEO_INTERRUPT	-10000，数据连接中断，通常情况是视频源存在问题或者数据格式不支持，比如音频不是AAC。

表 4-6 media_info_type 枚举

枚举成员	描述
MEDIA_INFO_UNKNOWN	1，未知信息。
MEDIA_INFO_STARTED_AS_NEXT	2，播放下一条。
MEDIA_INFO_VIDEO_RENDERING_START	3，视频开始准备中，准备渲染。
MEDIA_INFO_VIDEO_TRACK_LAGGING	700，视频日志跟踪。
MEDIA_INFO_BUFFERING_START	701，开始缓冲中，开始缓冲。
MEDIA_INFO_BUFFERING_END	702，缓冲结束。
MEDIA_INFO_NETWORK_BANDWIDTH	703，网络带宽，网速方面。
MEDIA_INFO_BAD_INTERLEAVING	800，音视频错乱传输，视频跟音频不同步。
MEDIA_INFO_NOT_SEEKABLE	801，直播，不可设置播放位置。
MEDIA_INFO_METADATA_UPDATE	802，一系列新的metedata可被使用。
MEDIA_INFO_TIMED_TEXT_ERROR	900，媒体信息计时文本错误。
MEDIA_INFO_UNSUPPORTED_SUBTITLE	901，不支持字幕。
MEDIA_INFO_SUBTITLE_TIMED_OUT	902，字幕超时。
MEDIA_INFO_VIDEO_ROTATION_CHANGED	10001，视频方向改变，视频选择信息。
MEDIA_INFO_AUDIO_RENDERING_START	10002，音频开始准备中。

5 iOS 播放器

5.1 开发前准备

软硬件环境配置要求

Mac电脑	• Mac® OS X® 10.10 (Yosemite) 或更高版本。 • 闪存：3GB RAM。 • 存储空间：至少2GB可用磁盘空间。
Xcode集成开发环境	Xcode 7.0版本或以上版本。
iOS终端设备	• iPhone：iPhone 5s及以后发布的设备。 • iPad：全部版本。
iOS操作系统	iOS7以上版本，推荐使用iOS8.0以上的版本。

SDK 集成

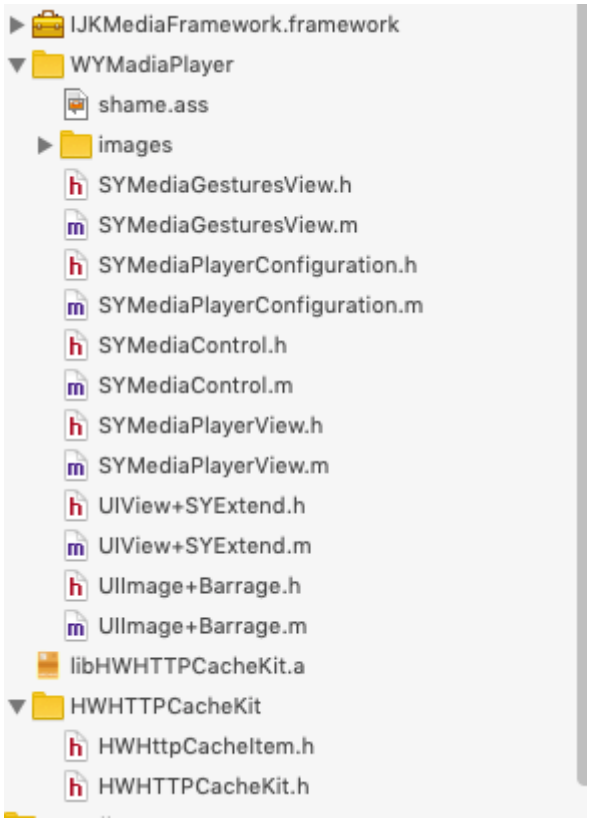
步骤1 解压**下载**的SDK压缩包。

步骤2 将“CloudVideo_iOS\VideoCloud\VideoCloud\Resource\NewHwPlayer”目录下的“IJKMediaFramwork.framwwork”文件拷贝到项目中。

若您需要实现视频边下载边播放或者实现缓存功能，则同时需要将“libHWHTTPCacheKit”拷贝到项目中。

步骤3 拷贝相关头文件和库文件到工程平级或者更深一级的目录下，如**图5-1**所示。

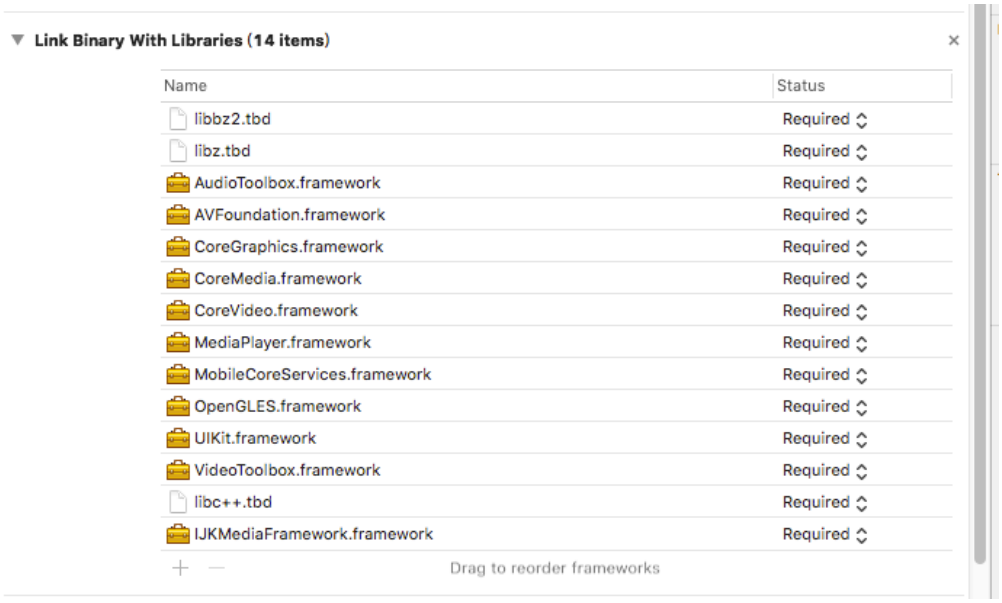
图 5-1 工程目录



步骤4 添加如下链接库，如图5-2所示。

- libbz2.tbd
- libz.tbd
- libc++.tbd
- AudioToolbox.framework
- VideoToolbox.framework
- AVFoundation.framework
- CoreGraphics.framework
- MediaPlayer.framework
- MobileCoreServices.framework
- OpenGLES.framework
- UIkit.framework

图 5-2 添加链接库



----结束

5.2 SDK 使用

实现播放功能

步骤1 根据APP需求设置options参数，设置规则请参考**IJKFFOptions类**。

```
IJKFFOptions *options = [IJKFFOptions optionsByDefault];
[options setPlayerOptionIntValue:30 forKey:@"max-fps"];
[options setPlayerOptionIntValue:30 forKey:@"r"];
//跳帧开关
[options setPlayerOptionIntValue:1 forKey:@"framedrop"];
[options setPlayerOptionIntValue:0 forKey:@"start-on-prepared"];
[options setPlayerOptionIntValue:0 forKey:@"http-detect-range-support"];
[options setPlayerOptionIntValue:48 forKey:@"skip_loop_filter"];
[options setPlayerOptionIntValue:0 forKey:@"packet-buffering"];
[options setPlayerOptionIntValue:2000000 forKey:@"analyzeduration"];
[options setPlayerOptionIntValue:25 forKey:@"min-frames"];
[options setPlayerOptionIntValue:1 forKey:@"start-on-prepared"];
[options setCodecOptionIntValue:8 forKey:@"skip_frame"];
[options setFormatOptionValue:@"nobuffer" forKey:@"fflags"];
[options setFormatOptionValue:@"8192" forKey:@"probsize"];
//自动转屏开关
[options setFormatOptionIntValue:0 forKey:@"auto_convert"];
//重连次数
[options setFormatOptionIntValue:1 forKey:@"reconnect"];
//开启硬解码
[options setPlayerOptionIntValue:1 forKey:@"videotoolbox"];
```

步骤2 初始化播放器。

```
self.player = [[IJKFFMoviePlayerController alloc] initWithContentURL:self.url withOptions:options];
//设置是否自动播放
self.player.shouldAutoplay = YES;
```

- **self.url**：播放地址。
- **options**：设置参数。

步骤3 设置播放区域。

```
self.playView = [[UIView alloc] initWithFrame:CGRectMake(0, 0, ScreenWidth, ScreenHeight)];
self.playView.backgroundColor = [UIColor blackColor];
[self.view addSubview:self.playView];
UIView *playingView = [self.player view];
playingView.frame = self.playView.bounds;
[self.playView insertSubview:playingView atIndex:1];
```

步骤4 开始播放。

```
[self.player setScalingMode:IJKMPMovieScalingModeFill];
[self installMovieNotificationObservers];
if(![self.player isPlaying]){
[self.player prepareToPlay];
 }
```

步骤5 暂停播放。

```
if ([self.player isPlaying]) {
      [self.player pause];
}
```

步骤6 执行播放。

```
if (![self.player isPlaying]) {
      [self.player play];
}
```

步骤7 倍速播放。

```
 (void)setRate:(float)rate{
   self.player.playbackRate = rate;
}
```

步骤8 停止播放。

```
if ([self.player isPlaying]) {
      [self.player stop];
}
```

步骤9 释放播放资源。

该步骤会停止播放器并释放播放器视图资源。

```
if (self.player) {
      [self.player shutdown];
      [self.player.view removeFromSuperview];
      self.player=nil;
   }
```

----结束

外挂字幕功能

iOS外挂字幕功能支持“.ass”和“.srt”格式的字幕文件，支持同时显示中文和英文字幕，支持随时调整字幕时间以实现字幕和视频的同步。

步骤1 初始化。

```
IJKSubtitleView *subtitleView = [[IjKSubtitleView alloc] initWithFrame:CGRectMake(0,0,375,40)];
[subtitleView configurationSubtitleViewWithContenPath:[[NSBundle main] pathForResource:@"shame"
ofType:@"ass"]];
/// 也可以在视频播放时改动
subtitleView.changeTime = 8;
[self.Play.view addsubView:subtitleView];
```

步骤2 链接播放器。

```
[subtitleView setPlay:self.delegatePlayer]
```

可以在播放器对象创建成功后再进行链接，若播放器传入为空，则会造成字幕显示出错或者无法显示。

步骤3 销毁字幕。

```
[subtitleView destroySubtitle];
```

每次结束播放时需要调用stop方法，并在stop方法中执行字幕销毁方法。

----结束

重连播放

iOS播放器SDK可以实现如下情况下的重连播放。

- 当主播由于网络等问题导致推流中断时，若在有效时间内重新推流，并且播放器端未播放新的直播流，则可以重新播放。
- 当观众观看直播时因网络原因中断时，若在有效时间内重新连接网络，则播放器可以恢复重新播放。

通过播放器的hook协议即可实现重连播放，当前重连功能暂只支持RTMP和HTTP协议。

```
NSString *url = @"rtmp://kitplay.hwcloudlive.com/live1/pushLiveSdkDemo";
    if ([url containsString:@"http"] || [url containsString:@"https"]) {
        url = [NSString stringWithFormat:@"ijkhttphook:%@",url];
    } else if ([url containsString:@"rtmp"] || [url containsString:@"rtsp"])
    {
        url = [NSString stringWithFormat:@"ijklivehook:%@",url];
    }
```

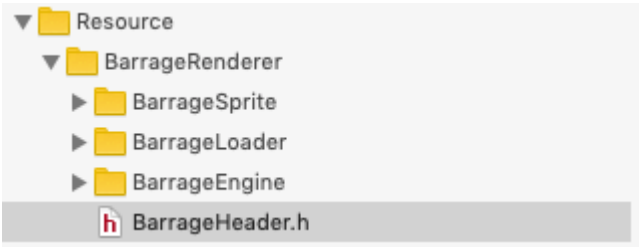
弹幕功能

iOS播放器SDK的弹幕功能是通过开源弹幕渲染库“BarrageRenderer”实现的。

步骤1 下载“BarrageRenderer”弹幕渲染库，

步骤2 将“BarrageRenderer”文件夹中的全部资源导入到Xcode项目中。

图 5-3 弹幕资源文件



步骤3 初始化。

```
    _renderer = [[BarrageRenderer alloc]init];
    _renderer.delegate = self;
    _renderer.redisplay = YES;
    [self.view addSubview:_renderer.view];
    [self.view sendSubviewToBack:_renderer.view];
    [self.renderer start];
```

步骤4 添加弹幕。

```
#pragma mark - 弹幕描述符生产方法
//生成精灵描述 - 延时文字弹幕
- (BarrageDescriptor *)walkTextSpriteDescriptorWithDelay:(NSTimeInterval)delay{
    BarrageDescriptor * descriptor = [[BarrageDescriptor alloc]init];
    descriptor.spriteName = NSStringFromClass([BarrageWalkTextSprite class]);
    descriptor.params[@"text"] = [NSString stringWithFormat:@"延时弹幕(延时%.0f秒)",delay];
    descriptor.params[@"textColor"] = [UIColor blueColor];
```



```
    descriptor.params[@"speed"] = @(100 * (double)random()/RAND_MAX+50);
    descriptor.params[@"direction"] = @(1);
    descriptor.params[@"delay"] = @(delay);
    return descriptor;
}
// 图文混排精灵弹幕 - 过场图文弹幕A
- (BarrageDescriptor *)walkImageTextSpriteDescriptorAWithDirection:(NSInteger)direction{
    BarrageDescriptor * descriptor = [[BarrageDescriptor alloc]init];
    descriptor.spriteName = NSStringFromClass([BarrageWalkTextSprite class]);
    descriptor.params[@"text"] = [NSString stringWithFormat:@"AA-图文混排/::B过场弹幕:%@",@"华为视频云
"];
    descriptor.params[@"textColor"] = [UIColor greenColor];
    descriptor.params[@"speed"] = @(100 * (double)random()/RAND_MAX+50);
    descriptor.params[@"direction"] = @(direction);
    return descriptor;
}
// 图文混排精灵弹幕 - 过场图文弹幕B
- (BarrageDescriptor *)walkImageTextSpriteDescriptorBWithDirection:(NSInteger)direction{
    BarrageDescriptor * descriptor = [[BarrageDescriptor alloc]init];
    descriptor.spriteName = NSStringFromClass([BarrageWalkTextSprite class]);
    descriptor.params[@"text"] = [NSString stringWithFormat:@"AA-图文混排/::B过场弹幕:%@",@"华为视频云
"];
    descriptor.params[@"textColor"] = [UIColor greenColor];
    descriptor.params[@"speed"] = @(100 * (double)random()/RAND_MAX+50);
    descriptor.params[@"direction"] = @(BarrageWalkDirectionL2R);    NSTextAttachment * attachment =
[[NSTextAttachment alloc]init];
    attachment.image = [[UIImage imageNamed:@"icon_cutout"]
    barrageImageScaleToSize:CGSizeMake(25.0f, 25.0f)];
    NSMutableAttributedString * attributed = [[NSMutableAttributedString alloc]initWithString:[NSString
stringWithFormat:@"BB-图文混排过场弹幕:%@",@"华为视频云"]];
    [attributed insertAttributedString:[NSAttributedString attributedStringWithAttachment:attachment]
atIndex:7];    descriptor.params[@"attributedText"] = attributed;
    descriptor.params[@"textColor"] = [UIColor magentaColor];
    descriptor.params[@"speed"] = @(100 * (double)random()/RAND_MAX+50);
    descriptor.params[@"direction"] = @(BarrageWalkDirectionL2R);
    descriptor.params[@"delay"] = @(direction);
    return descriptor;
}
  [self.view insertSubview:_renderer.view aboveSubview:self.player.view];
    NSInteger const number = 100;
    NSMutableArray * descriptors = [[NSMutableArray alloc]init];
    for (NSInteger i = 0; i < number; i++) {
        [descriptors addObject:[self walkTextSpriteDescriptorWithDelay:i*2+1]];
        [descriptors addObject:[self walkImageTextSpriteDescriptorBWithDirection:i*2]];
    }
    [_renderer load:descriptors];
```

步骤5 暂停弹幕。

```
[self.renderer pause];
```

步骤6 停止弹幕。

```
 [self.renderer stop];
```

----结束

播放器优化

- 丢帧处理

  默认音视频处理方案是视频同步音频，当CPU处理视频速度过慢时，会导致音频播放快于视频导致丢帧现象。

  优化方案：通过修改framedrop进行优化，在CPU无法正常处理视频时，丢弃视频非关键帧达到与音频同步的效果，ijkplayer默认为1，可根据实际需求进行修改。

  ```
  [options setPlayerOptionIntValue:0 forKey:@"framedrop"]; // 跳帧开关。
  ```

  如果CPU解码能力不足，可以设置成5，否则会引起音视频不同步，也可以通过设置跳帧达到倍速播放。

- 降码率

  如果使用硬编码，在网络环境较差的情况下，为了使直播画面更为流畅，可以实时改变硬编码率，进行丢帧处理，在丢帧的同时也可降低音频的码率。

  ```
  [options setOptionIntValue:1 forKey:@"videotoolbox" ofCategory:kIJKFFOptionCategoryPlayer]; // 开启硬编码videotoolbox（解码模式 0:软解码；1:硬解码）。
  ```

- 软硬编解码的选择

  表 5-1 软硬编解码说明

名称	描述	优点	缺点
软编解码	主要使用CPU进行编解码，通常情况下使用FFmpeg进行编码和解压音视频数据。	• 在不同的设备、系统版本中兼容性极高。 • 解码时，色彩还原度更高。 • 编解码过程可扩展性强。	CPU占用高，手机易发热，耗电量大。
硬编解码	主要使用非CPU进行编解码，如GPU等，通常情况下直接调用系统API进行音视频编解码处理。	系统占用少，执行效率高。	• 兼容性低，需根据硬件厂商和系统版本单独适配。 • 可控性比较差。

  - iOS硬件芯片为统一的videoToolBox解码后的NV12数据格式。
  - Android使用的mediacodec解码的数据格式跟手机硬件厂商有关，无法统一mediacodec解码出来的数据格式。

  综合以上情况，在推流方面，iOS系统和硬件设备统一性高，使用全硬编方案效果更好；Android因机型繁杂，支持程度不一，推荐4.3以上版本使用硬编。在播放解码方面，iOS硬解和软解支持性都较高，软解功耗更高，但是在部分细节方面表现较优，可控性强，具体视项目情况选择；Android推荐4.1版本以上使用硬解，4.1以下版本使用软解。

- 直播与点播建议设置参数如下所示。

  ```
  if (_isLive) {
  // Param for living
  [options setPlayerOptionIntValue:3000 forKey:@"max_cached_duration"];  // 最大缓存大小是3秒，可以根据实际需求进行修改
  [options setPlayerOptionIntValue:1 forKey:@"infbuf"];  // 无限读
  [options setPlayerOptionIntValue:0 forKey:@"packet-buffering"];  //  关闭播放器缓冲
  } else {
  // Param for playback
  [options setPlayerOptionIntValue:0 forKey:@"max_cached_duration"];
  [options setPlayerOptionIntValue:0 forKey:@"infbuf"];
  [options setPlayerOptionIntValue:1 forKey:@"packet-buffering"];
  }
  ```

- 累积延迟
  - RTMP到达CDN时存在4.0到5.0秒的延迟，可以通过设置IJKPlayer相关参数进行优化。

表 5-2 参数说明

参数	说明
skip_loop_filter	对所选帧进行跳过环路滤波。
skip_frame	编解码器丢弃所有非关键帧。

- 优化延迟问题还可以从推流端将编码器调低GOP，如1秒一个GOP，这样可以减小延迟时长，但会导致编码器压缩率降低，图像质量下降，并且会加大服务端的数据压力。
- 导致延迟还有一个主要原因是ffplay默认的帧率控制没有追帧策略，若网络发生抖动，延时会累加。优化策略就是调整帧率控制部分，保障流畅度和实时性。

5.3 接口参考

IJKMediaPlayback 类

表 5-3 接口说明

接口名	描述
IJKMPMediaPlaybackIsPreparedToPlayDidChangeNotification	播放状态的改变代替MPMoviePlayerContentPreloadDidFinishNotification。
IJKMPMoviePlayerScalingModeDidChangeNotification	缩放比例的改变。
IJKMPMoviePlayerPlaybackDidFinishNotification IJKMPMoviePlayerPlaybackDidFinishReasonUserInfoKey	NSNumber (IJKMPMovieFinishReason) 当电影播放结束或用户退出播放时调用。
IJKMPMoviePlayerPlaybackStateDidChangeNotification	用户改变播放状态改变时调用。
IJKMPMoviePlayerLoadStateDidChangeNotification	当网络加载状态发生变化时。
IJKMPMoviePlayerIsAirPlayVideoActiveDidChangeNotification	当视频通过AirPlay开始播放视频或结束时调用Movie Property Notifications属性相关的同时声明。
IJKMPMovieNaturalSizeAvailableNotification	在执行prepareToPlay时开始异步确定影片属性，当相关属性变为有效可用时调用该通知。
IJKMPMoviePlayerVideoDecoderOpenNotification	编译器打开通知。

接口名	描述
IJKMPMoviePlayerFirstVideoFrameRenderedNotification	视频第一帧时通知。
IJKMPMoviePlayerFirstAudioFrameRenderedNotification	音频第一段时通知。

表 5-4 IJKMediaPlayback 方法说明

方法	描述
- (void)prepareToPlay	准备播放。
- (void)play	播放。
- (void)pause	暂停。
- (void)stop	停止播放。
- (BOOL)isPlaying	获取播放状态，是否正在播放。
- (void)setPauseInBackground: (BOOL)pause	后台暂停。
- (void)shutdown	这句要写在viewDidDisappear中，否则可能会造成vc不会被释放。
- (UIImage *)thumbnailImageAtCurrentTime	获取当前时间的封面帧图片。

表 5-5 IJKMediaPlayback 属性说明

属性	描述
@property (nonatomic, readonly) UIView *view	用于显示视频播放的view，调用次view
@property (nonatomic) NSTimeInterval currentPlaybackTime	当前播放的时间点
@property(nonatomic, readonly) NSTimeInterval duration	总时长
@property(nonatomic, readonly) NSTimeInterval playableDuration	可播放时长
@property(nonatomic, readonly) NSInteger bufferingProgress	缓冲进度
@property(nonatomic, readonly) BOOL isPreparedToPlay	准备播放

属性	描述
@property(nonatomic, readonly) IJKMPMoviePlaybackState playbackState	播放终止状态枚举
@property(nonatomic, readonly) IJKMPMovieLoadState loadState	加载状态枚举
@property(nonatomic, readonly) int64_t numberOfBytesTransferred	传输字节数
@property(nonatomic, readonly) CGSize naturalSize	视频原始显示View size
@property(nonatomic) IJKMPMovieScalingMode scalingMode	视频尺寸模式
@property(nonatomic) BOOL shouldAutoplay	需要自动播放
@property (nonatomic) BOOL allowsMediaAirPlay	支持AirPlay媒体
@property (nonatomic) BOOL isDanmakuMediaAirPlay	支持弹幕AirPlay媒体
@property (nonatomic, readonly) BOOL airPlayMediaActive	AirPlay是否活跃
@property (nonatomic) float playbackRate	返回音频/视频的当前播放速度

表 5-6 IJKMediaUrlOpenDelegate 方法说明

方法	描述
- (id)initWithUrl:(NSString *)url	初始化URL。
event:(IJKMediaEvent)event	设置URL打开协议，具体设置请参考**表 5-13**。
segmentIndex:(int)segmentIndex	设置分片个数。
retryCounter:(int)retryCounter	设置重试次数。

表 5-7 IJKMediaUrlOpenDelegate 属性说明

属性	描述
@property(nonatomic, readonly) int IJKMediaEvent event	URL协议event。

属性	描述
@property(nonatomic, readonly) int segmentIndex	分片次数。
@property(nonatomic, readonly) int retryCounter	重试次数。
@property(nonatomic, retain) NSString *url	播放URL。
@property(nonatomic) int error	错误提示，发生错误该属性为负。
@property(nonatomic, getter=isHandled) BOOL handled	如果URL发生改变，该数值变为Yes。
@property(nonatomic, getter=isUrlChanged) BOOL urlChanged	通过改变URL，设置为YES。

表 5-8 IJKMPMovieLoadState 类

枚举成员	描述
IJKMPMovieLoadStateUnknown	0，未知状态。
IJKMPMovieLoadStatePlayable	1 << 0，视频未完成全部缓存，但已缓存的数据可以进行播放。
IJKMPMovieLoadStatePlaythroughOK	1 << 1，完成缓存，如果设置了自动播放（shouldAutoPlay为Yes），完成缓存后会自动播放。
IJKMPMovieLoadStateStalled	1 << 2，数据缓存已经停止，播放将暂停。

表 5-9 IJKMPMoviePlaybackState 类

枚举成员	描述
IJKMPMoviePlaybackStateStopped	播放停止。
IJKMPMoviePlaybackStatePlaying	开始播放。
IJKMPMoviePlaybackStatePaused	暂停播放。
IJKMPMoviePlaybackStateInterrupted	播放间断。
IJKMPMoviePlaybackStateSeekingForward	播放快进。
IJKMPMoviePlaybackStateSeekingBackward	播放后退。

表 5-10 IJKMPMovieScalingMode 类

枚举成员	描述
IJKMPMovieScalingModeNone	无缩放比例。
IJKMPMovieScalingModeAspectFit	尺寸比例不变填满屏幕为止。
IJKMPMovieScalingModeAspectFill	尺寸比例不变填满屏幕，可能造成内容缺少。
IJKMPMovieScalingModeFill	尺寸比例变形也会填满屏幕。

表 5-11 IJKMPMovieFinishReason 类

枚举成员	描述
IJKMPMovieFinishReasonPlaybackEnded	完成原因：播放结束。
IJKMPMovieFinishReasonPlaybackError	完成原因：播放出现错误。
IJKMPMovieFinishReasonUserExited	完成原因：出现用户人为退出。

表 5-12 IJKMPMovieTimeOption 类

枚举成员	描述
IJKMPMovieTimeOptionNearestKeyFrame	获取缩略图的时间点数组。
IJKMPMovieTimeOptionExact	准确时间。

表 5-13 IJKMediaEvent 类

分类	枚举成员	描述
Notify Events	IJKMediaEvent_WillHttpOpen	1，HTTP通道打开事件attr：url。
	IJKMediaEvent_DidHttpOpen	2，TCP通道打开事件attr：url、error、http_code。
	IJKMediaEvent_WillHttpSeek	3，HTTP通道的seek事件attr：url、offset。
	IJKMediaEvent_DidHttpSeek	4，HTTP通道的seek事件attr：url、offset、error、http_code。
Control Message	IJKMediaCtrl_WillTcpOpen	0x20001，TCP控制通道打开IJKMediaUrlOpenData：no args。

分类	枚举成员	描述
	IJKMediaCtrl_DidTcpOpen	0x20002，TCP控制通道seek IJKMediaUrlOpenData：error、family、ip、port、fd。
	IJKMediaCtrl_WillHttpOpen	0x20003，HTTP控制通道打开 IJKMediaUrlOpenData：url、segmentIndex、retryCounter。
	IJKMediaCtrl_WillLiveOpen	0x20005，Live控制通道打开 IJKMediaUrlOpenData：url、retryCounter。
	IJKMediaCtrl_WillConcatSegmentOpen	0x20007，ConcatSegment控制通道打开IJKMediaUrlOpenData：url、segmentIndex、retryCounter。

IJKMediaModule 类

表 5-14 属性说明

属性	描述
@property(atomic, getter=isAppIdleTimerDisabled)BOOL appIdleTimerDisabled	APP运行时是否锁屏，若您不需要在运行程序时锁屏，则设置为YES。
@property(atomic, getter=isMediaModuleIdleTimerDisabled) BOOL mediaModuleIdleTimerDisabled	处于非播放状态是否锁屏，若您不需要在非播放状态时锁屏，则设置为YES。

IJKFFOptions 类

表 5-15 IJKAVDiscard 类

枚举内容	描述
IJK_AVDISCARD_NONE	-16，不丢帧。
IJK_AVDISCARD_DEFAULT	0，如果包大小为0，则抛弃无效的包。
IJK_AVDISCARD_NONREF	8，抛弃非参考帧（I帧）。
IJK_AVDISCARD_BIDIR	16，抛弃B帧。
IJK_AVDISCARD_NONKEY	32，抛弃除关键帧以外的帧，比如B，P帧。
IJK_AVDISCARD_ALL	48，抛弃所有的帧

表 5-16 方法和属性

分类	参数	描述	示例
kIJKFFOptionCategoryPlayer	r	帧速率（fps）。 可以根据实际需求进行修改，但为保证音画同步建议帧速率设定为15或者29.97。	[options setPlayerOptionIntValue:29.97 forKey:@"r"];
kIJKFFOptionCategoryPlayer	framedrop	跳帧开关。 取值范围：[-1,120] ● 0：表示关闭，默认值。 ● 其它值：表示丢帧程度 建议设置为5。	[options setPlayerOptionIntValue:0 forKey:@"framedrop"];
kIJKFFOptionCategoryPlayer	vol	设置音量的大小。 256为标准音量，若需要设置成两倍音量时则输入512，依此类推。	[options setPlayerOptionIntValue:512 forKey:@"vol"];
kIJKFFOptionCategoryPlayer	max-fps	最大fps。	[options setPlayerOptionIntValue:30 forKey:@"max-fps"];
kIJKFFOptionCategoryPlayer	packet-buffering	关闭播放器缓冲。 若频繁卡顿，则保留缓冲区。不设置则默认为1。	[options setPlayerOptionIntValue:0 forKey:@"packet-buffering"];
kIJKFFOptionCategoryPlayer	videotoolbox	开启硬编码。 默认为0：软解。	[options setPlayerOptionIntValue:1 forKey:@"videotoolbox"];
kIJKFFOptionCategoryPlayer	an	静音设置。	[option setPlayerOptionValue:@"1" forKey:@"an"];
kIJKFFOptionCategoryPlayer	vn	是否有视频。	[option setPlayerOptionValue:@"1" forKey:@"vn"];
kIJKFFOptionCategoryPlayer	nodisp	是否禁止图像显示(只输出音频)。	[options setPlayerOptionIntValue:1 forKey:@"nodisp"];

分类	参数	描述	示例
kIJKFFOptionCategoryPlayer	videotoolbox-max-frame-width	指定最大宽度。	[options setPlayerOptionIntValue:960 forKey:@"videotoolbox-max-frame-width"];
kIJKFFOptionCategoryPlayer	max_cached_duration	设置缓存大小。 可根据实际需求进行修改。	[options setPlayerOptionIntValue:3000 forKey:@"max_cached_duration"]; [options setPlayerOptionIntValue:0 forKey:@"max_cached_duration"];
kIJKFFOptionCategoryPlayer	infbuf	设置是否无限读。	[options setPlayerOptionIntValue:1 forKey:@"infbuf"]; [options setPlayerOptionIntValue:0 forKey:@"infbuf"];
kIJKFFOptionCategoryPlayer	packet-buffering	是否关闭播放器缓冲。	[options setPlayerOptionIntValue:0 forKey:@"packet-buffering"]; [options setPlayerOptionIntValue:1 forKey:@"packet-buffering"];
kIJKFFOptionCategoryPlayer	start-on-prepared	开始准备。	[options setPlayerOptionIntValue:0 forKey:@"start-on-prepared"];
kIJKFFOptionCategoryPlayer	overlay-format	fourcc叠加格式。	[options setPlayerOptionIntValue:@"fcc-_es2" forKey:@"overlay-format"];
kIJKFFOptionCategoryPlayer	video-pictq-size	最大图片队列帧数。	[options setPlayerOptionIntValue:3 forKey:@"video-pictq-size"];
kIJKFFOptionCategoryPlayer	min-frames	最小帧停止预读。	[options setPlayerOptionIntValue:25 forKey:@"min-frames"];

分类	参数	描述	示例
kIJKFFOptionCategoryCodec	skip_loop_filter	开启环路滤波，具体参数设置请参考表5-15。 • 0较48清晰，但解码开销大。 • 48清晰度低，解码开销小。	[options setCodecOptionIntValue:IJK_AVzDISCARD_DEFAULT forKey:@"skip_loop_filter"];
kIJKFFOptionCategoryCodec	skip_frame	累积延迟。 • RTMP到达CDN时存在4.0到5.0秒的延迟，可以通过设置对应参数进行优化。 • ffplay默认的帧率控制没有追帧策略，若网络发生抖动，延时会累加。 优化策略：调整帧率控制部分，保障流畅度和实时性。	[options setCodecOptionIntValue:IJK_AVDISCARD_DEFAULT forKey:@"skip_frame"];
kIJKFFOptionCategoryFormat	rtsp_transport	如果使用RTSP协议，可以优先选用tcp（默认udp）。	[options setFormatOptionValue:@"tcp" forKey:@"rtsp_transport"];
kIJKFFOptionCategoryFormat	probesize	播放前的探测Size，默认是1M，若设置为小于1M会更快播出画面。	[options setFormatOptionIntValue:1024 * 16 forKey:@"probesize"];
kIJKFFOptionCategoryFormat	analyzeduration	播放前的探测时间。	[options setFormatOptionIntValue:50000 forKey:@"analyzeduration"];
kIJKFFOptionCategoryFormat	auto_convert	自动转屏开关。	[options setFormatOptionIntValue:0 forKey:@"auto_convert"];
kIJKFFOptionCategoryFormat	reconnect	重连次数。	[options setFormatOptionIntValue:1 forKey:@"reconnect"];
kIJKFFOptionCategoryFormat	timeout	超时时间，“timeout”参数只对“http”设置有效。若使用“RTMP”设置“timeout”，ijkplayer内部会忽略“timeout”参数。	[options setFormatOptionIntValue:30 * 1000 * 1000 forKey:@"timeout"];

分类	参数	描述	示例
无	showHudView	显示加载指示器视图。	options.showHudView = NO;

IJKFFMoviePlayerController 类

表 5-17 LogLevel 日志等级枚举说明

枚举内容	描述
k_IJK_LOG_UNKNOWN	0，未知
k_IJK_LOG_DEFAULT	1，默认
k_IJK_LOG_VERBOSE	2，详细
k_IJK_LOG_DEBUG	3，调试
k_IJK_LOG_INFO	4，详情信息
k_IJK_LOG_WARN	5，警告
k_IJK_LOG_ERROR	6，错误
k_IJK_LOG_FATAL	7，致命
k_IJK_LOG_SILENT	8，静默、无日志

表 5-18 方法说明

方法	描述

- (id)initWithContentURL:(NSURL *)aUrl
withOptions:(IJKFFOptions *)options
	获取视频路径，创建播放器。
	//本地视频路径
	NSString* localFilePath=[[NSBundle mainBundle]pathForResource:@"测试视频" ofType:@"mp4"];
	NSURL *localVideoUrl = [NSURL fileURLWithPath:localFilePath]; //网络视频路径
	NSString *webVideoPath = @"https://651.cdn-vod.huaweicloud.com/asset/58ae22459ce00e735807bf5621cfc170/play_video/db50c8f66bd83bc4c7cd573eb8cabfbc_H.264_1920X1080_HEAACV1_3000.mp4";
	NSURL *webVideoUrl = [NSURL URLWithString:webVideoPath];
	_viewController = [[IJKFFMoviePlayerController alloc]initWithContentURL:webVideoUrl];

- (id)initWithContentURLString:(NSString *)aUrlString withOptions:(IJKFFOptions *)options	获取视频路径。
- (void)prepareToPlay	播放准备。

方法	描述
- (void)play	播放。
- (void)pause	暂停。
- (void)stop	停止播放。
- (BOOL)isPlaying	是否正在播放中。
- (void)setPauseInBackground: (BOOL)pause	后台暂停。
- (BOOL)isVideoToolboxOpen	视频工具栏是否开启。
+ (void)setLogReport: (BOOL)preferLogReport	日志报告输出。
+ (void)setLogLevel: (IJKLogLevel)logLevel	日志报告等级。
+ (BOOL)checkIfFFmpegVersionMatch: (BOOL)showAlert	检查版本是否匹配。
+ (BOOL)checkIfPlayerVersionMatch: (BOOL)showAlert	检查版本是否匹配。

表 5-19 属性说明

属性	描述
@property(nonatomic, readonly) CGFloat fpsInMeta	fps率。
@property(nonatomic, readonly) CGFloat fpsAtOutput	fps输出值。