媒体云播放器

Android SDK 用户手册

---基于 Jar 包及 so 动态库方案

V1.4s

发布日期： 2013年8月8日

百度开发者中心

目录

第1章　概述

百度媒体云播放器 Android SDK（以下简称“播放器 SDK”）是百度官方推出的 Android 平台使用的软件开发工具包（SDK），为 Android 开发者提供简单、快捷的接口，帮助开发者实现 Android 平台上的媒体播放应用开发。

播放器 SDK 内嵌百度自主研发的 T5 播放内核，对目前主流的本地和网络媒体都提供了良好的功能支持，弥补了系统播放器在媒体支持格式上的不足，并在兼容性、稳定性和响应速度上有明显的提高。

播放器 SDK 提供了多种层面的调用方式，开发者可根据自己的需求定制化开发播放界面，也可使用默认播放界面实现快速开发。

同时播放器 SDK 提供了跨越电视、PC、平板或智能手机的多媒体互动播放的能力，目前支持 Android 端的 DLNA(Digital Living Network Alliance)控制器，可控制市面上兼容 DLNANA 协议的电视、机顶盒产品和大量智能设备，支持音频、视频、图片的跨设备播放，能实现基本的遥控器功能，提供丰富便捷的互动体验。

播放器 Android SDK 提供了两种集成模式：

- 共享播放引擎方案

  该方案下，各播放应用使用同一播放引擎，应用包本身容量可以大大减小。该方案 SDK 版本号为*.*，如 1.4。

- Jar 包及 so 动态库集成方案

  该方案下，各播放应用使用独享的播放引擎，免去了安装 apk 播放引擎的过程。该方案 SDK 版本号为*.*s，如 1.4s。

同版本号下，两种方案 SDK 提供的功能是一致。

本手册为 Jar 包及 so 动态库集成方案的用户手册，共享播放引擎方案请到百度开发中心网站（developer.baidu.com）的媒体云服务下载相关开发包及用户手册。

播放器 SDK 的完整下载包中包含 demo、doc、lib 和用户指南四个部分， 解压后的目录结构如下所示：

- demo： 主要存放 2 个 Android 示例工程，可以帮助用户了解如何使用该 SDK。其中 sample1 是同时调用 BVideoView 和 BMediaController 相关的示例；sample2 则是只调用 BVideoView 的相关示例。DLNAControllerDemo 则是实现多屏互动的示例。
- doc： 主要存放播放器 SDK 相关接口参考文档，可离线查看 index.html 获取 API 的具体说明。
- lib： 播放器 SDK 开发包，包括 jar 包、so 及相关资源文件。解压后得到 cyberplayer-sdk.jar，res 目录，cyberplayer-cores 目录；cyberplayer-sdk.jar 是我们的 java 层实现，在您的工程中进行引用，res 目录存放的 BMediaController 引用的资源文件，cyberplayer-cores 目录存放的是各个手机类型的 so 压缩包，解压任何一个包获得 libs 目录，将 libs 目录合并到您工程跟目录的 libs 下即可。

- SDK 用户手册

第2章　阅读对象

本文档面向所有使用该 SDK 的开发人员、测试人员、合作伙伴以及对此感兴趣的其他用户，要求读者具有一定的 Android 编程经验。

第3章　媒体云 SDK 功能说明

播放器 SDK 以开发者为中心，以高效创建媒体播放应用为目标，具有以下的特色功能：

1. 低门槛、高灵活度实现播放功能

   利用 SDK 提供的 API 接口轻松创建专业级播放应用；UI 界面可自由定制。

2. 轻松实现多屏互动

   利用 SDK 提供的 API 接口，轻松实现基于 DLNA 的多屏互动应用，实现图片、视频在不同手机、平板、电视间分享和控制。

3. 流媒体格式完美支持

   跨 Android 版本(2.2 以上)完美支持 HTTP Streaming 及 HTTP Live Streaming(HLS，M3U8)流媒体协议。

4. 无缝支持百度个人云存储（PCS）视频转码接口

   在播放百度个人云存储中的视频时，可根据设备计算能力及带宽的不同，调用百度云转码服务，智能转码实现移动端的流畅播放。

5. 本地全媒体格式支持

   支持目前所有主流的媒体格式（mp4，avi，wmv，flv，mkv，mov，rmvb 等）。

6. 智能统计

   开发者可以通过 web 管理界面，及时有效的查看播放器 SDK 给出的各种统计及分析结果。

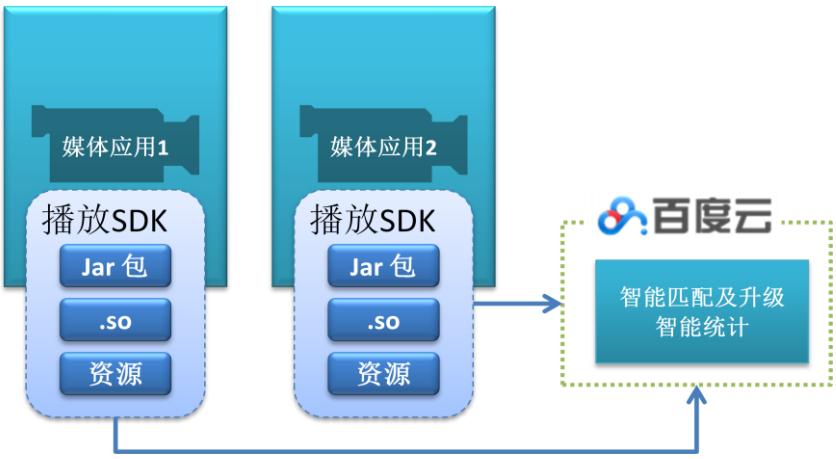
第4章　媒体云 SDK 使用说明

4.1　集成 SDK

4.1.1　整体框架

开发者需要将播放器 SDK 集成到应用中方可使用，包括 Jar 包，底层库及相关资源文件。每个应用需要独立集成 Jar 包及动态库。如果您的应用场景中多个应用希望共享一套播放引擎，请采用媒体云提供的共享播放引擎集成方案。

设计框架结构如下图所示：



图表 1 Jar 包及动态库集成方式示意图

4.1.2 移动设备 CPU 类型及特征

移动设备 CPU 根据架构不同主要有 ARM 系列，x86 系列等。ARM 系列又分 armv5,armv6,armv7；并且分别支持不同的特性，主要有 vfp,vfpv3,neon 等。为了取得最佳的播放性能， 播放内核针对各种 CPU 及其特性做了相应优化，并且形成了各种版本的 so 动态库。高版本手机可兼容使用低版本库，但是无法使用本机具备的硬件加速特性，影响性能表现。所以建议选择同手机硬件对应的库匹配使用。 SDK 中提供了 8 种不同 CPU 的动态库，具体请参考表格 2 中的内容。

库类型 / 手机类型	Armv5	Armv5 -vfp	Armv6	Armv6- vfp	Armv7- vfp	Armv7- vfpv3	Armv7- neon	X86
Armv5	√							
Armv5-vfp	√	√						
Armv6	√		√					
Armv6-vfp	√	√	√	√				
Armv7-vfp	√	√	√	√	√			
Armv7-vfpv3	√	√	√	√	√	√		
Armv7-neon	√	√	√	√	√	√	√	
X86	√	√	√	√	√	√	√	√

图表 2 动态库兼容性列表

注意：动态库同 CPU 架构和指令集直接关联，选择不匹配的动态库，可能造成应用无法播放或者崩溃。x86 处理器能够以二进制转换的方式支持 Arm 指令集，但会对性能造成较大影响，建议在 x86 手机上使用原生的 x86 动态库。

4.1.3 集成同 CPU 匹配的 so 动态库

选择同 CPU 匹配的动态库，会带给视频应用最佳的播放体验。为方便开发者完成匹配工作，播放 SDK

提供了 CPU 类型检测以及运行时在线匹配并下载 so 动态库的接口函数。以下给出三种动态库集成策略，：

1. 应用程序仅集成 Jar 包，在首次启动应用时，调用 SDK 的 API 在线请求媒体云服务器，获取同 CPU 相匹配的 so 动态库。相关的 API 调用请具体参考 6.3.1 小节。
2. 应用程序集成 Jar 包，缺省内置单一版本的 so 动态库，并在程序首次运行时进行匹配检测，发现不匹配时再通过在线下载动态库，完成动态库更替，API 调用参考 6.3.1 小节。目前市场上主流机型的 CPU 采用 armv7-neon 架构，建议在程序中默认内置 armv7-neon 的动态库，但一些早期采用 armv5, armv6 CPU 架构的手机无法直接运行，需要在线匹配。如果希望获取最大程度的兼容性，可内置 armv5 架构的动态库，但主流机型均无法充分发挥硬件性能。
3. 应用程序集成 Jar 包并内置多版本的 so 动态库，程序首次运行时完成匹配，选择加载最适合的 so 动态库。 适用于无网络的应用场景，但会明显增大应用的程序包大小。

4.1.4 集成 So 动态库注意事项

考虑到安卓系统对 So 动态库使用上的限制，以及使用安卓系统硬件的多样性，请开发者关注如下几点：

1. 下载底层动态库压缩包后，解压的 so 动态库不能放置在 sdcard 中让播放内核加载，此操作受限于系统对动态库加载的限制。如果是被 root 过的工程机可能会加载成功，但是仍然不建议这么做。
2. 如果程序已经成功加载动态库并完成视频播放，之后又下载新的库并设置加载路径后再进行播放，加载新的库不会成功，仍然会使用旧的库，这受限于 Android 系统机制，在这种情况下必须重启程序才能生效。
3. 获取 CPU 类型可能出现 unknown 的情况，原因可能是 CPU 不是 arm 系列或者 x86 系列，或者一些手机硬件不规范导致不能识别，这种情况只能提示用户不兼容或者尝试使用内置的版本库去播放。如果遇到此类情况可以报问题给我们，提供描述具体手机型号等信息，我们会尽量协助解决。

4.2 使用 SDK 实现媒体播放功能

使用 SDK，可实现以下两种播放界面的：

- 使用 BVideoView 及 BMediaController 快速实现
- 使用 BVideoView，自己实现播放控制界面

4.2.1 使用 BVideoView 及 BMediaController 快速实现

为满足开发者构建复杂、个性媒体播放应用，开发定制化播放器的需求，播放器 SDK 为开发者提供 BVideoView 和 BMediaController 两个类接口，其开发方式与基于 Android 原生播放器开发相同，具体请参考 API 调用中的相关说明。

其中，BVideoView 为媒体播放显示提供支持；BMediaController 则为媒体播放控制提供支持，其中包括“播放\暂停”键及其它两个自定义键，可实现类似“上一部”及“下一部”功能，当然也可以实现你自

己的想实现的。如果注册了两个相应的 listener，自定义键会显示在 controller 中，否则不会显示。在 controller 中也显示了当前播放时间及视频总时间，并有 seekbar，可以对当前播放进行 seek 操作。

4.2.2 使用 BVideoView，自己实现播放控制界面

在 BMediaController 不满足你界面风格或者交互的时候你可以只使用 BVideoView 来进行播放画面的显示，自己在程序中实现播放控制界面，通过 BVideoView 获取相关播放信息及注册相关 listener 来实现。

4.3 使用 SDK 实现多屏互动

SDK 除了支持视频播放外，还提供了基于 DLNA 控制器的多屏互动支持，来实现手机，平板电脑，PC，以及智能电视（机顶盒）之间的内容的共享与互动。整个库包含一套完整的 SDK，内含丰富的 API，涵盖 DLNA 激活与停止，设备的发现，资源文件的设定，播放控制，状态查询，事件订阅及出错通知，帮助开发者轻松创建功能强大的媒体应用，为用户提供优质家庭影音及娱乐新体验。

主要包含以下特点：

7. 支持图片，音频，视频等多种格式的媒体文件
8. 良好的兼容性，支持主流电视（Sony）或机顶盒（小米盒子，Letv，快播大屏幕）
9. 功能强大，支持拖动，音量控制等功能，完全可以代替遥控器
10. 延时小，反应迅速；体积小，运行时资源损耗小
11. 使用方便，只需导入 jar 包即可，无额外库依赖
12. 同步调用，简化编程；异步通知，实时可靠

SDK 目前仅开放了 DLNA 的控制器功能，使用 API 可以发现并控制网络中的其他 DLNA 设备，如果希望将智能设备支持 DLNA 协议并能够被其他的控制器控制，请关注后续的 SDK 更新。

第5章 开发前准备

5.1 运行环境

- Android 2.2 及以上的所有系统
- 支持的硬件 CPU 目前覆盖：

  ARM v5，ARM v6，ARM v7，及 Intel X86

5.2 参数申请及权限开通

开发者需要使用百度账号登录百度开发者中心注册成为百度开发者并创建应用，方可获取应用 ID、对应的 API Key（即：ak）及 Secret Key（即：sk）等信息。具体信息，请参考百度开发者中心上的“创建应用”的相关介绍。

SDK 认证时必须传入 ak 及 sk 参数。为保证安全性，在设置 sk 时请仅输入前 16 个字节。

第6章　使用 SDK 开发播放器应用

6.1　添加播放器 SDK 到 App 工程

请参考以下步骤，将播放器 SDK 添加到 App 工程中：

1. 创建一个 Android 工程；
2. 将 SDK 中的 Jar 包添加到 App 工程的 libs 目录下，并在工程属性中设置依赖此 Jar 包。；
3. 选取同 CPU 类型匹配的 so 动态库压缩包，并解压到工程目录的 libs 目录。
4. 将我们发布的 res 文件夹合并到工程目录的 res 目录（此步可选，如果你确定不会用到 BMediaController，可以不用拷贝资源）。

6.2　权限声明

在您的 Android App 的 AndroidManifest.xml 中声明如下权限：

```
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_SETTINGS" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

6.3　调用 API

下面介绍如何调用播放器 SDK 中已封装的 API 完成各项操作：

6.3.1　检查底层动态库的匹配

1. **获取当前系统 CPU 类型**

   通过 VersManager.getCurrentSystemCpuTypeAndFeature()获取当前 CPU 类型，此接口会发起网络请求，并在服务器端实现 CPU 类型的决策，以保证匹配策略对新硬件的快速支持和在线更新能力。该 API 为异步操作，开发者需实现接口回调并以参数传入。

   如果当前 CPU 类型与你内置的底层库类型不一致时，你可能需要进行库的重新匹配。

2. **获取当前版本对应 CPU 类型底层库下载地址**

   通过 VersionManager.getDownloadForCurrentVersion()获取到同参数中所输入的 CPU 类型对应的动态库下载地址。

3. **下载底层库的 zip 包**

   通过标准的 http 协议下载动态库 zip 包，在 http 头里面会有“Content-MD5”字段，此字段是 zip 包 md5 字符串，便于下载后对下载文件的校验。

4. **解压 zip 包获取到底层库**

底层库包含 libcyberplayer.so 和 libcyberplayer-core.so。

5. **使用解压后的 so 动态库**

将 so 拷贝或直接解压至应用程序文件目录（/data/data/your_app_package_name/files）。在播放前通过 BVideoView.setNativeLibsDirectory()设置 so 所在的目录路径。

6.3.2 实现媒体播放功能

如 4.2 节所述，使用 SDK 实现媒体播放功能有两种方式。本节即针对上述两种方式的具体实现进行详细说明。

6.3.2.1 使用 BVideoView 及 BMediaController 快速实现播放

1. 创建 BVideoView 及 BMediaController

通过以下方式之一创建 BVideoView 及 BMediaController：

- 直接在布局文件中创建
- 在源码中动态创建，然后添加到页面中

方式 1：直接在布局文件中创建

示例如下：

```
…
      <RelativeLayout
       android:id="@+id/videoview_holder"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
      android:layout_centerInParent="true"
      <com.baidu.cyberplayer.core.BvideoView
            Android:id=”@id/videoview”
            Android:layout_width=”fill_parent”
            Android:layout_height=”fill_parent”
      />
      />
…
      <RelativeLayout
       android:id="@+id/videoview_holder"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
      android:layout_centerInParent="true"
      <com.baidu.cyberplayer.core.BMediaController
            Android:id=”@id/controllerbar”
            Android:layout_width=”fill_parent”
            Android:layout_height=”fill_parent”
      />
      />
…
```

方式 2： 在源码中动态创建，然后添加到页面中

示例如下：

```
BVideoView mVV;
BMediaController mVVCtl;
…
mViewHolder = (RelativeLayout) findViewById(R.id.videoview_holder);
mControllerHolder = (LinearLayout) findViewById(R.id.controlbar_holder);

mVV = new BVideoView(MediaControllerPlayingActivity.this);
mVVCtl = new BMediaController(MediaControllerPlayingActivity.this);
LinearLayout.LayoutParams param = new LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.FILL_PARENT,
                LinearLayout.LayoutParams.FILL_PARENT);
mViewHolder.addView(mVV, param);
mControllerHolder.addView(mVVCtl, param);
```

2. 关联 BVideoView 与 BMediaController 实现播放

首先，如果需要指定底层库的加载位置，在播放前调用 BVideoView.setNativeLisbDirectory()，此调用只需执行一次即可，不用每次播放前都调用。

其次，可注册 listener 监听播放过程，以便应用程序根据监听结果进行相应处理。示例如下：

```
mVV.setOnPreparedListener();
mVV.setOnCompletionListener();
mVV.setOnErrorListener();
mVV.setOnInfoListener();
mVV.setOnPlayingBufferCacheListener();
```

然后，关联 BVideoView 与 BMediaController，开始播放。示例如下：

```
mVV.setMediaController(mVVCtl);
mVV.setDecodeMode(BVideoView.DECODE_SW); //可选择软解模式或硬解模式
mVV.setVideoPath(path);
mVV.showCacheInfo(true/false);
mVV.start();
```

其中 showCacheInfo()是设定当缓冲时是否在 videoview 上显示缓冲状态和缓冲百分比，默认是会显示。setDecodeMode()是设定软硬解模式。默认是软解模式，开发者可根据 Android 系统默认支持的格式情况来决定，如果是默认支持的格式（如 MP4、3GP）建议打开硬解开关以节省功耗、提高播放性能。

6.3.2.2 使用 BVideoView，自己实现播放控制界面

BVideoView 的使用跟跟 6.3.2.1 一样。

实现播放控制界面需要注意：

- 在调用 BVideoView.start()后需要等待 OnPreparedListener.onPrepared()来表示视频准备播放就绪，这个过程你可能需要在界面做一些等待信息或提示。
- 在 OnPreparedListener.onPrepared()后你可以通过 BVideoView.getCurrentPosition(),BVideoView.getDuration()获取到当前播放时间及视频总时长来显示到界面及设置进度条。进度条拖动时可以通过 BVideoView.seekTo()来对视频进行 seek 操作。
- 执行 BVideoView.stopPlayback（）停止视频，此操作是异步操作，需要等待 OnCompletionListener.onCOmpletion()，这表示播放真正结束，才能进行下一次播放。

第7章 使用 SDK 开发多屏互动应用

SDK 开放了 DLNA 的控制器功能，实现多屏互动。

7.1 添加控制端 SDK 到 App 工程

参见 6.1 即可，权限声明参考 6.2 即可

7.2 调用 API

使用 DLNA 功能, API 的调用分为以下几步：

- 传入 AK，SK 进行初始化和权限验证
- 启动 DLNA 服务
- 获取 Render（电视等显示设备）列表
- 指定某个 Render 与之关联
- 指定待播放资源的 URL
- 播放控制阶段，开始，暂停，拖动，调解音量；
- 停止 DLNA 服务

其他可选步骤：

- 播放前获取 Render 能力，支持哪些格式和协议
- 播放过程中，主动获取 Render 状态信息
- 订阅 Render 状态，避免主动轮询
- 出错监听

注意： IDLNAServiceProvider 是对外服务的唯一接口，在执行完初始化操作后，必须先调用 addActionCallback 注册回调， API 调用根据是否为耗时操作分为同步调用和异步调用；同步调用立即返回结果；异步调用立即返回，通过已注册对应的回调获取执行结果，由于回调不是在 UI 线程中执行，请勿在回调中执行 UI 相关操作；如果操作失败，还可以通过对应回调查看错误码和错误消息详情。API 文档中标明了具体 API 是同步还是异步操作。

7.2.1 获取 DLNA 服务实例

首先通过 DLNAProviderFactory 的 getInstance 方法获取服务实例：

```
IDLNAServiceProvider DLNAProviderFactory::getInstance(Context
context);
```

7.2.2 初始化 SDK

传入 AK，SK 调用以下接口进行初始化

说明：AK,SK 的获取参见 5.2

```
void initialize(String accessKey, String secretKey);
```

7.2.3 注册 ActionCallback

通过注册 ActionCallback，得到异步操作的结果

说明：该步骤是必须的，而且必须在所有 DLNA 操作之前进行执行

```
boolean addActionCallback(DLNAActionListener);
```

7.2.4 启动 DLNA 服务

调用 IDLNAServiceProvider 的以下 API 即可启动底层服务，执行结果通过 DLNAActionlistener 中的 onEnableDLNA 回调返回，APP 运行过程中只需要启动一次该服务即可。

说明：异步调用,除非系统底层网络服务不可用，否则 onEnableDLNA 都将返回 true

```
void enableDLNA ();
```

7.2.5 获取 Render 列表

调用 IDLNAServiceProvider 的以下 API 即可得到 Render 列表，如果当前网络没有 Render，则返回 null：

```
List<String> getRenderList();
```

7.2.6 关联到某个 Render

调用 IDLNAServiceProvider 的以下 API 即可关联到指定的 Render，接下来就可以对 Render 进行控制和设定了，通过对应回调获取关联结果，随后对统一 render 进行的操作都无需再次关联：

```
void selectRenderDevice(String devName);
```

7.2.7 设置待播放资源的 URL

调用 IDLNAServiceProvider 的以下 API 即可设置媒体资源的 URL，DLNAActionlistener 中的 onSetMediaURI 将会返回设置结果：

```
void setMediaURI(String mediaURI);
```

7.2.8 IDLNAServiceProvider 调用 API 进行播放控制

调用 IDLNAServiceProvider 的以下 API 进行播放控制，由于交互对象是网络中的其他终端设备，控制和交互过程存在网络延时和设备响应延时，即 API 调用返回后，被控制设备可能会出现延后一段时间作出响应的情况。

被控制的终端所处的状态会影响 API 调用是否成功，譬如只有在播放状态下才能去进行 Seek 操作。所有的控制操作结果均通过 DLNAActionlistener 中的对应回调函数返回。

播放：

```
void play( );
```

暂停：

```
void pause();
```

停止播放：

```
void stop();
```

拖动：position：xx:xx:xx 格式

```
void seek(position);
```

设置音量：volume:音量百分比

```
void setVolume(int volume);
```

设置是否静音：

```
void setMute(Boolean isMute);
```

7.2.9 播放过程中获取被控设备状态

调用 IDLNAServiceProvider 的以下 API 获取被控设备相关状态，如果操作失败，请结合错误码进行分析；通常由设备异常或者网络中断引起，请及时检查设备或网络状态。

获取音量：请求音量，获取音量后会返给注册的 DLNAActionlistener::onGetVolume;

```
void getVolume();
```

获取 Mute 状态：请求获取被控设备是否处于静音状态，获取后会将结果返给注册的 DLNAActionlistener::onGetMute;

```
void getMuteStat();
```

获取媒体文件播放时长：立即返回播放总时长

说明：可能返回 00:00:00，尝试重新获取；推荐注册 DeviceEventListener，duration 发生变化时底层负责通知上层最新的 duration 值，如果一直拿不到正常值，可以尝试主动获取

```
String getMediaDuration();
```

获取当前的播放进度：立即返回当前的播放位置

说明：可能返回 00:00:00，忽略并重新获取，推荐注册 DeviceEventListener，底层同时上层最新的 Position 值

```
String getMediaPosition() ;
```

获取 Render 的当前状态：立即返回 Render 的当前状态，推荐注册 DeviceEventListener，当远端 Render 设备状态变化时底层通知上层应用

```
String getRenderState()
```

7.2.10 停止 DLNA 服务

要结束 DLNA 的服务，请调用以下 API，便于系统进行资源回收及线程关闭，异步操作

```
void disableDLNA();
```

7.2.11 订阅被控设备状态变动通知

关联设备成功后，根据以下几步，可订阅被控设备的状态变动通知，避免主动轮询：

1. 实现 DeviceEventListener 接口
2. 添加自定义监听器

```
boolean addEventListener(DeviceEventListener listener);
```

说明：如果订阅底层事件更新，请保持数据来源的单一性，防止上层 App 和底层通知同时操作某个状态值导致的异常

7.2.12 获取被控设备的多媒体格式支持能力集

作为 Debug 工具使用，检查是否是超出设备播放能力造成的播放失败：

```
String getSupportedProtocols();
```

第8章 API 说明

有关 API 的详细说明，请查看所附 SDK 包 doc 目录下的《API 参考文档》。

第9章 媒体云 SDK 升级

播放器 SDK 可通过百度开发者中心媒体云相关帮助文档处下载获取更新。

第10章 播放信息统计

媒体云服务为开发者提供了媒体应用播放的相关统计信息，开发者可以通过查看统计信息了解应用的使用现状。

进入媒体云服务管理控制台步骤如下：

- 登录进入百度开发者中心的“管理中心”；
- 点击使用媒体云服务的应用，进入应用基本信息页；
- 点击左侧边栏中的“云平台 > 媒体服务 > 媒体播放”，即可进入播放器 SDK 统计查看页面。

第11章 联系我们

如果以上信息无法帮助您解决在开发中遇到的具体问题，请通过以下方式联系我们：

邮箱: dev_support@baidu.com

百度工程师会在第一时间回复您。

第12章 文档变更历史

版本号	发布日期	描述
1.3s	2013.7.17	基于媒体云播放器 SDK1.3 发布 Jar 包及 so 方案，并形成此方案文档。