5G+超高清视频

设计方案

目录

1 方案概述

近年来，随着 5G 网络的迅猛发展， 4K、8K 极高清视频成为 5G 最佳应用场景，同时以 5G 手机、OTT 智能盒子、互联网电视、AR/VR 为主各种智能设备的普及也为 4K、8K 极高清视频的播放能力提供了硬件算力基础。目前 4K、8K 极高清业务尚在发展早期，面临着内容少、制作成本高的问题，随着播放设备的普及回馈内容制作，高质量、低成本的 4K、8K 极高清视频会越来越多。但同时随之而来的问题就是面对 4K、8K 极高清视频的网络分发则需要传统十倍的数据流量，这对于分发体系而言就存在巨大的压力和高昂的成本，通过运用闲置带宽和存储资源的 5G CPE、MEC 边缘计算网络则可很好的解决此类问题。此次 5G+超高清项目主要要求播放器能够适配指定的各种终端、操作系统和芯片平台，在满足媒体文件和网络媒体流基本播放需求的基础上，提供基于 5G CPE、MEC 边缘计算网络的 4K\8K 超高清播放。

根据当前超高清播放器项目的需求，我们提出了一套具有广泛适应性的专业媒体播放器解决方案。该播放器能提供对应应用进行调用，包含多种媒体流、多种视频解码的功能模块，并通过不同类型

的 API 接口，让用户在 native 层实现高质量跨平台播放，确保播放器与服务器端的互操作性，以便用户迅速完成集成/开发。

该播放器是一种多媒体播放器开发套件，可在连接的设备（包括手机，平板电脑，台式机，机顶盒和智能电视）上跨平台传送和播放内容。支持企业级和消费者级应用程序的开发，以满足高质量视频/音频传输和回放的最新趋势。

该播放器基于模块化架构，可提供最大的可移植性和可伸缩性。其通用的应用程序编程接口（API）支持跨多个平台的重用和可扩展部署。模块化体系结构还加速了功能丰富的开发。同时该播放器可以快速，轻松地集成到内容交付应用程序，第三方 DRM 模块或自定义协议中。视频和音频后处理支持实现了诸如隐藏式字幕和字幕渲染以及高级音频处理等功能。

该播放器工具包支持一系列针对最流行格式的高度优化的软件多媒体编解码器，并支持多种媒体服务器上的多种流协议。该播放器建立在经过市场验证的技术之上，该技术目前为超过 2 亿个移动设备提供播放能力。

2 方案设计

2.1 设计要点

2.1.1 分布式存储分发

本项目分布式存储分发是使用 IPFS 协议，IPFS 协议是一种内容可寻址、版本化、点对点超媒体的分布式存储、传输协议。IPFS 有八层子协议栈，从上至下为身份、网络、路由、交换、对象、文件、命名、应用，每个协议栈各司其职，又互相搭配。身份层和路由层可以对等节点身份信息的生成以及路由规则是通过 Kademlia 协议生成制定，KAD 协议实质是构建了一个分布式松散 Hash 表，简称 DHT，每个加入这个 DHT 网络的人都要生成自己的身份信息，然后才能通过这个身份信息去负责存储这个网络里的资源信息和其他成员的联系信息，本项目则是将 DHT 与本项目的用户 ID 绑定。网络层比较核心，使用的 LibP2P 可以支持任意传输层协议。NAT 技术能让内网中的设备共用同一个外网 IP。交换层，是类似 BT 下载，并创建中心服务器，当服务器登记用户请求资源时，让请求同样资源的用户形成一个小集群 swarm，在这里分享数据。IPFS 团队把 BitTorrent 进行了创新，叫作 Bitswap，它增加了信用和帐单体系来激励节点去

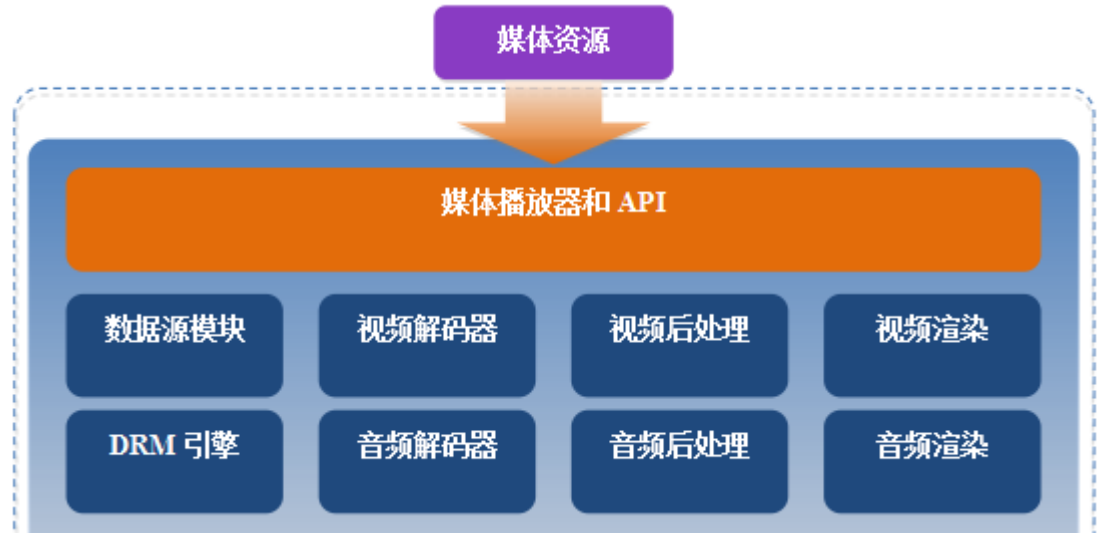
分享，用户在 Bitswap 里增加数据会增加信用分，分享得越多信用分越高。如果用户只去检索数据而不存数据，信用分会越来越低，其它节点会在嵌入连接时优先选择信用分高的。对象层和文件层理的是 IPFS 上 80%的数据结构，大部分数据对象都是以 MerkleDag 的结构存在，这为内容寻址和去重提供了便利。文件层是一个新的数据结构，和 DAG 并列，采用 Git 一样的数据结构来支持版本快照。命名层具有自我验证的特性（当其他用户获取该对象时，使用指纹公钥进行验签，即验证所用的公钥是否与 NodeId 匹配，这验证了用户发布对象的真实性，同时也获取到了可变状态），并且加入了 IPNS 这个巧妙的设计来使得加密后的 DAG 对象名可定义，增强可阅读性。应用层，我们可以利用它类似 CDN 的功能，在成本很低的带宽下，去获得想要的数据，从而提升整个应用程序的效率。

本项目分布式存储分发的服务则可通过类库的编译运行在 Linux、Windows、MacOS、Unix、FreeBSD 等诸多系统，从而可集成在 5G CPE、MEC 等 5G 边缘网络服务中。

2.1.2 播放能力框架设计

本项目是一款适用于多个平台和多个硬件设备的内容播放，能提供各种应用进行调用，多种媒体流、多种视频解码、渲染，并通过不同类型的 API 接口，让用户在 native 层实现高质量跨平台播放。

在底层的播放器分为三个层次：



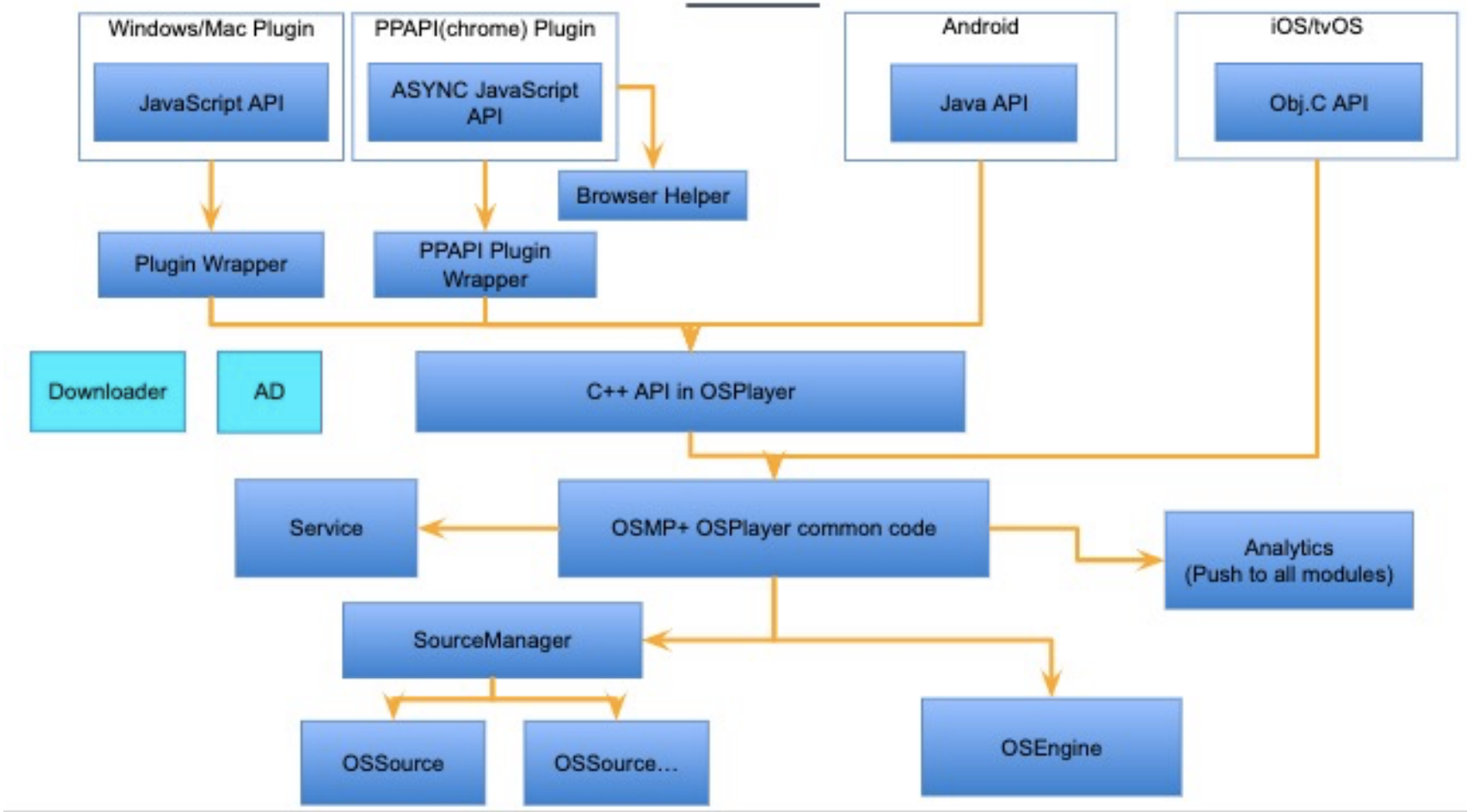
1. 上层包括多个 API 层组成：Android 的 Java API 层，iOS/tvOS 的 Obj.C API 层，Windows/MAC Plugin 的 JavaScript API 层，PPAPI(Chrome) Plugin 的 ASYNC JavaScript API 层和 HTML5 的 API 层。
2. 中间层，即多媒体框架层包含和上层互联的 C++ API、播放控制、码流自适应控制、数据源模块，DRM 引擎，音视频的渲

染，音视频的后处理等多个模块组成。

3. 底层包含了各种编解码格式的软编解码库，支持硬件设备的硬编解码功能等。

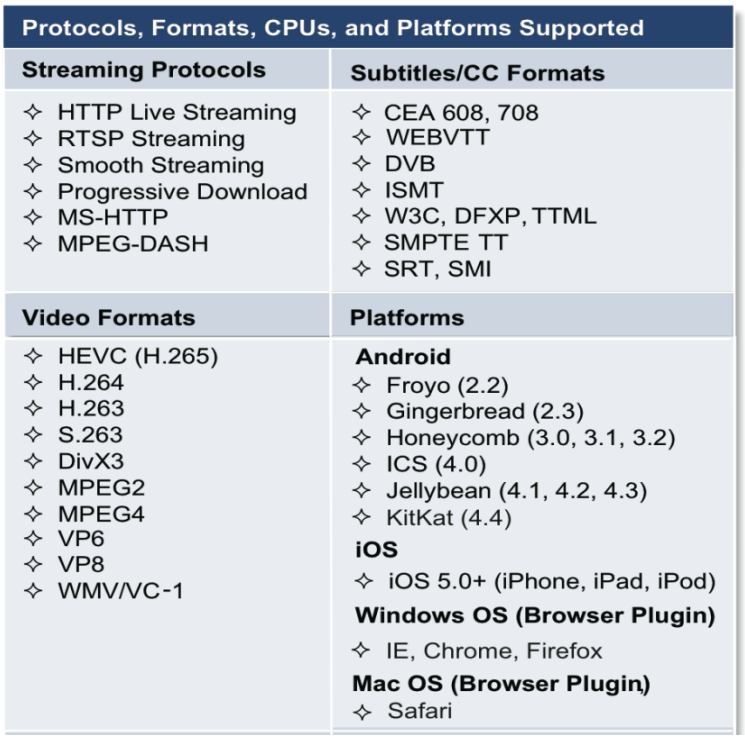
结构如下所示：

Architecture



2.1.3 支持格式

本项目能够支持主流的各种编码格式、封装格式、协议格式等，满足多种业务的运营需求。



Protocols, Formats, CPUs, and Platforms Supported

Streaming Protocols	Subtitles/CC Formats
✧ HTTP Live Streaming ✧ RTSP Streaming ✧ Smooth Streaming ✧ Progressive Download ✧ MS-HTTP ✧ MPEG-DASH	✧ CEA 608, 708 ✧ WEBVTT ✧ DVB ✧ ISMT ✧ W3C, DFXP, TTML ✧ SMPTE TT ✧ SRT, SMI
Video Formats	**Platforms**
✧ HEVC (H.265) ✧ H.264 ✧ H.263 ✧ S.263 ✧ DivX3 ✧ MPEG2 ✧ MPEG4 ✧ VP6 ✧ VP8 ✧ WMV/VC-1	**Android** ✧ Froyo (2.2) ✧ Gingerbread (2.3) ✧ Honeycomb (3.0, 3.1, 3.2) ✧ ICS (4.0) ✧ Jellybean (4.1, 4.2, 4.3) ✧ KitKat (4.4) **iOS** ✧ iOS 5.0+ (iPhone, iPad, iPod) **Windows OS (Browser Plugin)** ✧ IE, Chrome, Firefox **Mac OS (Browser Plugin)** ✧ Safari

1. 支持.ts、.mp4、.avi、.mkv、.flv 等封装各种的视频文件播放；
2. 支持 MPEG2、MPEG4、H.263、H.264、HEVC(H.265)、DivX3 等视频编码格式和 AAC、MP3、Dolby AC3／EAC3 格式的音频软解码；
3. 支持基于标准 HLS（HTTP Live Streaming）网络流媒体协议的视频播放，遵循 Apple 公司的 HLS 标准协议；对于 HLS 能够提供很好的容错能力，并且提供 API 进行容错参数的变更设置；
4. 支持基于标准 RTSP 网络流媒体协议的视频播放，遵循

RFC236/2327 标准；

5. 支持 RTSP 网络流的时延长短配置；
6. 支持基于标准 HTTP 网络流媒体协议的视频播放，支持 HTTP 网络流的容错及重连配置；
7. 支持基于标准 MPEG－DASH 网络流媒体协议的视频播放；
8. 支持基于微软提出的 Smooth Streaming 网络流媒体协议的视频播放；
9. 支持基于 ARM 构架 NEON 指令集优化，支持 X86 构架的软解码。

这些在后续章节会详细介绍。

2.1.4 稳定性

1. 项目能够支持各种基础播放功能，包括基本的解码播放和控制；
2. 在解码能力上具有世界级的专业水准，能够极好的支持本地媒体文件或指定网络媒体流的高质量播放；
3. 支持音视频自动合成、播放启动、停止、暂停、快进、快退,音量大小调节等常用操作，在播放停止时能够释放占用的资源；；
4. 在视频文件的播放过程中，支持从任意指定的播放位置开始播放，并支持任意时间点随机跳转操作；
5. 支持音视频基本信息的获取，可以通过接口的形式传递给相关的业务应用进行必要的查看；

6. 在播放器屏幕的设置方面能够提供很好的功能支撑业务的需求。支持设置播放器屏幕位置和大小，支持对屏幕亮度的调整，支持屏幕放大缩小功能，支持播放器屏幕 Pan&Scan 与 LetterBox 功能；
7. 支持设置播放器外观风格；
8. HLS，DASH，SS 支持带宽自适应功能，可以根据网络的情况来实时调整播放的码流以提高用户流畅播放的体验

2.2 播放能力 SDK 功能

2.2.1 Android 平台下的渲染选项

播放器在 Android 上支持以下三个渲染选项（也称为渲染类型），以便在需要时实现更好的播放性能：

- Native C: 自 Android N 起，该选项将被淘汰；
- Native windows: 播放器默认使用此选项；
- Open GL: 启用 enableCardBoardVideo 或 enableSphericalVideo 时必须选择此选项。

2.2.2 缩放模式

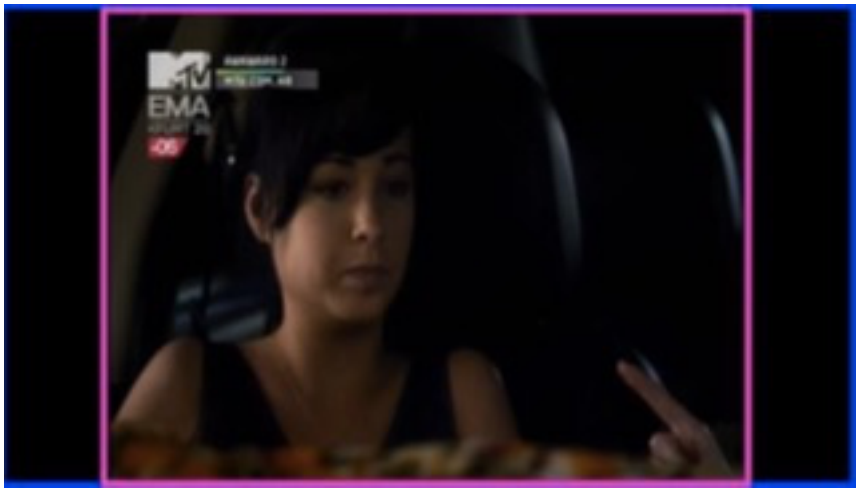
支持以下缩放模式来控制视频的呈现：

- Letterbox 模式
- Pan& Scan 模式
- Fit to Window 模式

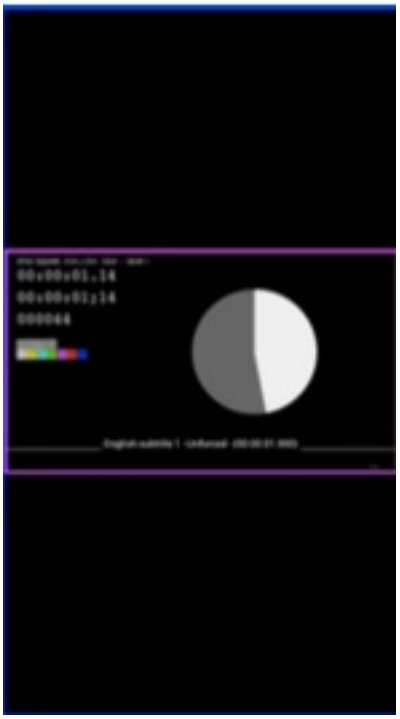
- Original Size 模式
- Zoom In 模式

2.2.2.1 Letterbox 模式

Letterbox 模式将视频的高度或宽度与显示区域匹配，同时保持其原始纵横比取决于显示区域的方向和大小。下图显示了一个示例，其中将 4:3 视频的高度更改为与 16:9 显示区域的高度匹配，从而在左侧和右侧产生黑条。



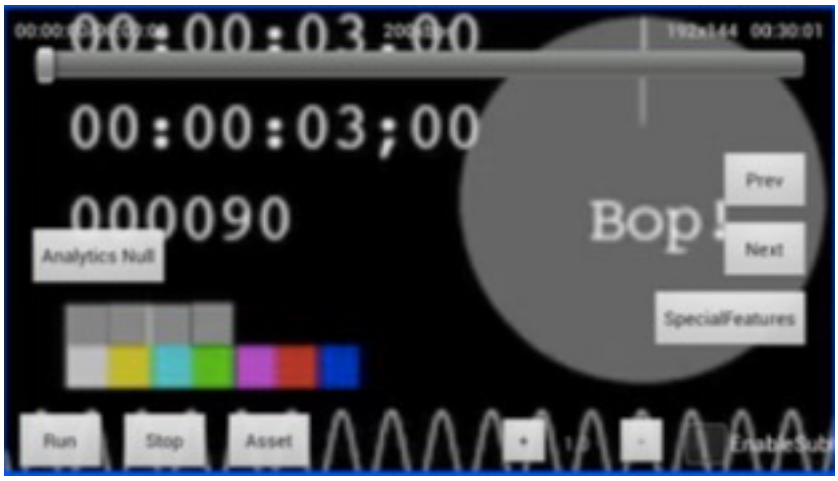
下图显示了一个示例，其中视频的宽度被调整为与旋转的 16:9 显示区域匹配，从而导致居中视频内容上方和下方出现黑条。



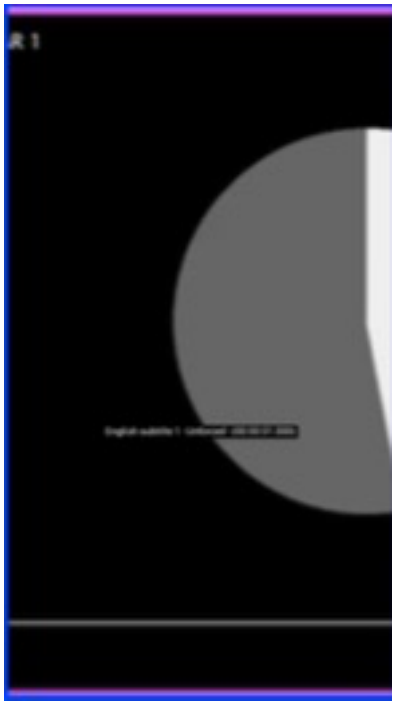
2.2.2.2 Pan&Scan 模式

“Pan&Scan”模式将视频的高度或宽度与显示区域匹配，同时根据显示区域的方向和大小保持其原始纵横比。与信箱模式相反，“Pan&Scan”模式从显示区域删除了任何黑条，从而有效地剪切了部分视频内容。

缩放在匹配方向的 16:9 显示区域上呈现的 4:3 视频，使其宽度与显示区域的宽度匹配，同时保持其宽高比。这导致视频感觉整个显示区域以及视频的一部分在顶部和底部被剪切，如下图所示。

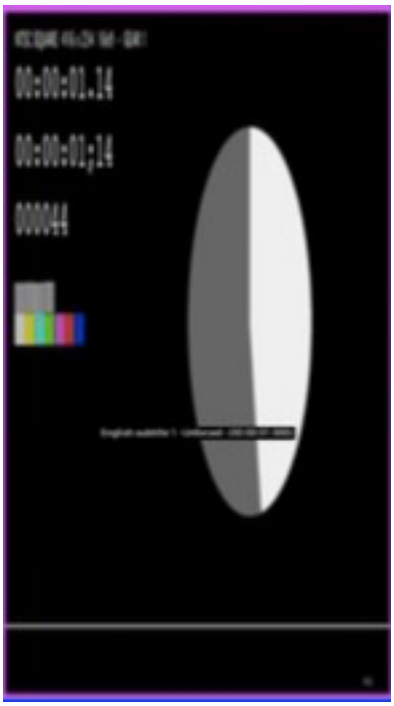


下图显示了以 Pan&Scan 模式在不同方向的 16:9 显示区域中渲染的 4:3 视频的示例，因此导致视频的高度与显示区域的高度匹配，并且在左侧和右侧剪切了视频。



2.2.2.3 Fit to Window 模式

无论视频的长宽比和大小如何，都将对其进行调整大小以填充整个显示区域，如下图所示。



2.2.2.4 Original Size 模式

原始大小模式将以原始大小和长宽比呈现视频，而不管显示区域的大小和长宽比如何，如下图所示。

2.2.2.5 Zoom In 模式

放大模式在视频中指定一个矩形区域以填充显示区域，同时保留原始视频纵横比，如下图所示。指定的矩形边界不能超出视频实际大小。

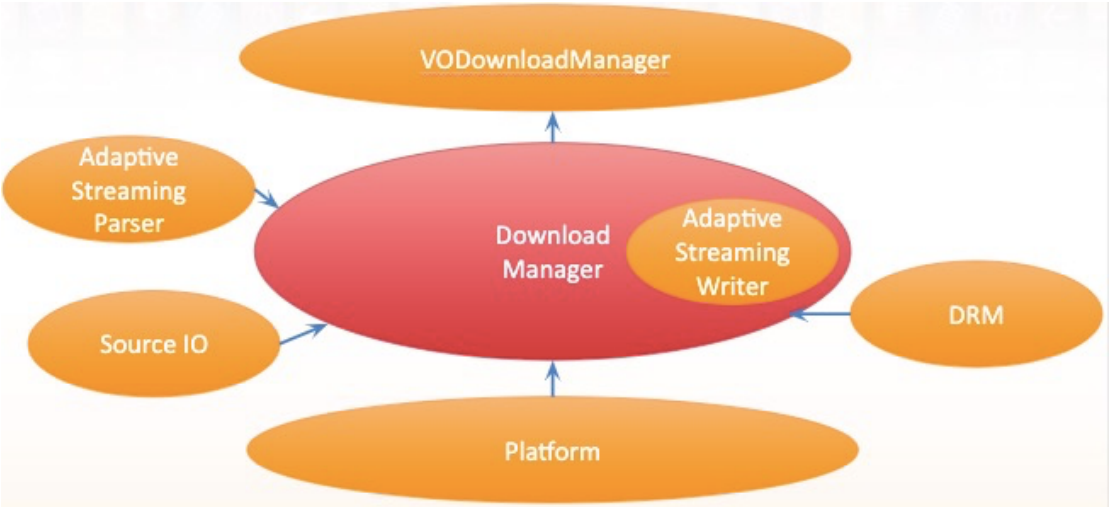
2.2.3 下载管理模块

下载管理模块是播放器功能，用于下载内容以供离线查看。下载管理模块支持以下功能：

- 在播放或仅下载的同时将内容下载到本地存储。如果下载管理模块已打开，则在开始播放时将提示“Select Asset”对话框；
- 支持后台下载；
- 选择要下载的视频比特率，音频轨道和字幕轨道。如果打开了下载管理器功能，则将提示您在“Select Asset”对话框中仅选择一个视频比特率，音频轨道或字幕轨道进行下载；

- 检查 DRM 密钥到期；
- 报告并显示下载进度；
- 从最后一个断点继续下载；
- 报告由于连接失败或设备存储已满而导致的错误；
- 开始，停止，暂停，继续和从应用程序中删除下载。

当前的下载管理模块支持与 DRM 结合使用的 HLS/DASH/Smooth Streaming VOD 流，结构如下图所示。

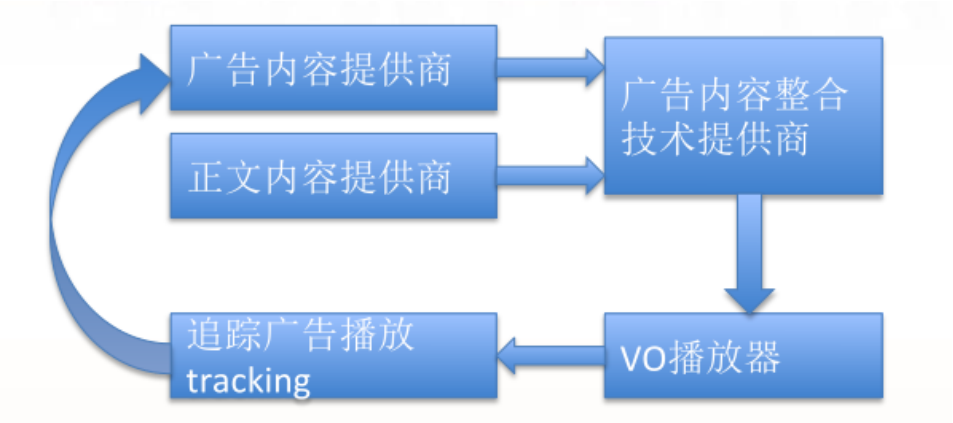


2.2.4 视频广告插播支持

本播放器能够支持广告业务的拓展，在各种视频节目播放的过程中进行视频类广告的无缝拼接插入。视频广告插播支持在正片视频中进行前插、中插和后插的方式来展示广告。为了能够节省带宽，避免多次下载，播放器还支持对广告资源的本地缓存；同时，播放器还支持广告展示剩余时间的获取和广告展示的计数统计。

一般一个广告插入系统需要以下五个要素组成 ：

1. 正文内容提供商；
2. 广告内容提供商；
3. 提供广告内容整合技术提供商；
4. 播放正文和广告的播放器；
5. 追踪广告播放的技术提供商。



- 广告商会把播放需求预先告知广告整合商 ；
- 整合商会把对应的广告按照对应的时机做事先的安排，插入正文 ；
- 播放器按照这个安排切换播放正文和广告；
- 播放完成后播放器或者第三方会把广告播放追踪信息告知广告商。

本项目播放器支持多种形式的广告插播。

（一）基于点播的双播放器实现介绍

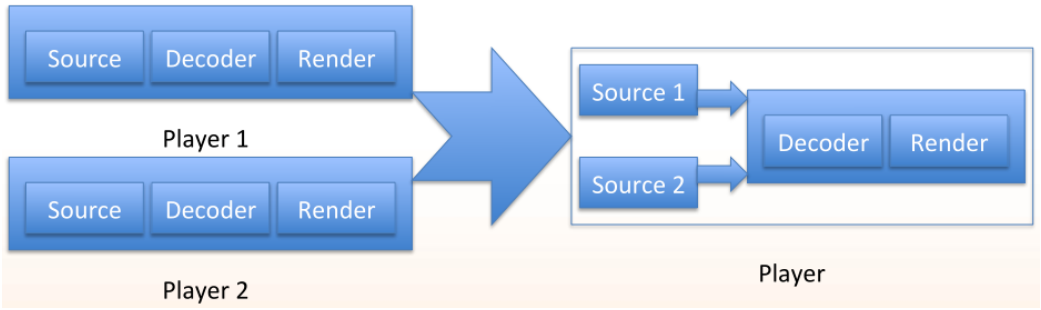
如果正片流是 Progress Download（HTTP）或者 HTTP Live

Streaming（HLS），广告流可以为任意格式的文件（ts，MP4 等）的 Progress Download（HTTP）的流。从而实现 PD＋PD 或者 HLS＋PD 的组合。

广告内容整合商可以把广告插入正文的任意位置（头部，尾部，中部）。播放器支持多线程多实例应用，可以用两个播放器，一个控制正文，另一个控制广告。当播放正文的时候第二个播放器处于暂停状态。当需要播放广告的时候可以暂停当前的播放器，随后激活第二个播放器用来播放广告。在播放过程中，如果广告支持 VASC 模式，播放器可以追踪并发送报告给广告服务器。

（二）单播放器两路 source 实现

由于双播放器的实现对 APP 的控制来说比较复杂，并且对系统资源有较多消耗，所以可以用单播放器去实现双路流的流畅切换。一个播放器主要由多媒体框架，协议解析器，解码器以及渲染模块来组成，除了协议解析模块以外的模块都可以被复用。因此可以把此模块独立出来，其他的模块就可以被复用。此方法的优点是尽量降低内存的使用和不占用更多的带宽，只需要一个播放器来实现，不同流之间可以更流畅的切换。 使用双播放器实现广告插播时，不建议同时调用硬件解码器，但是采用单播放器实现广告插播没有此限制。



2.2.5 直播回看（DVR）

DVR 功能使观众可以暂停，播放和回看实时事件。如果观看者在初始直播时间之后开始观看事件，并且不想错过任何重要内容，则可以使他们从较早的位置开始。播放器会检测 HLS 播放列表是否为 DVR 流，并调整控制栏上的可视元素以使直播流可以暂停或倒退到过去的某个点。

在播放器在播放期间，将进度条滚动到目标位置，会出现以下三种情况：

- 小于 DVR 的最小位置时，播放器将从最早的存储历史记录中重播。
- 等于或大于 DVR 最大位置，播放器将返回实时模式。
- 在 DVR 最小值和 DVR 最大值位置之间，播放器将从相应的历史记录中重播。

2.2.6 I-Frame 缩略图预览

播放器带有 HLS 流 V2 模块的 Android 和 iOS 上提供此功能。带有 I 帧流的 HLS/VOD 内容支持缩略图预览，本地文件（例如

MP4）也支持它。通过此功能，应用程序可以通过为播放器提供开始位置，结束位置和间隔来请求缩略图。JPEG 图片可用时，播放器返回回调。然后，应用程序可以在 UI 上插入 JPEG 缩略图。

有关更多详细信息，请参阅 API 参考手册中的以下方法和事件：

- isThumbnailAvailable()
- requestThumbnails()
- VO_OSMP_CB_THUMBNAILS_REQUEST_UPDATE
- VO_OSMP_CB_THUMBNAILS_REQUEST_FINISH
- VO_OSMP_THUMBNAILS_PREFERENCE

2.2.7 外部扩展 VTT 文件缩略图预览

在时间滑块上设置新位置时，播放器支持将缩略图预览作为外部 VTT 文件进行显示，以显示叠加预览图像。叠加缩略图作为外部 VTT 文件加载到播放器。W3C VTT 格式是一种基本的文本格式，结合了定时和缩略图。这些缩略图以 JPG 之类的基本图像格式存储。

缩略图预览还支持多种 JPEG 大小写格式。可以将多个图合并为一个 W3C 媒体片段，然后将各个拇指映射到正确的时间。

有关更多详细信息，请参阅 API 参考手册中的以下方法和事件：

- setThumbnailURI ()
- requestThumbnails()
- VO_OSMP_CB_THUMBNAILS_REQUEST_URI_UPDATE
- VO_OSMP_CB_THUMBNAILS_REQUEST_FINISH

2.2.8 iOS 平台上的 Airplay

AirPlay 是专有协议栈和套件，允许在音频，视频，设备屏幕和照片的设备之间以及相关数据之间进行无线流传输。

在 iOS 上启用 AirPlay 后，您可以播放/停止，暂停，挂起/继续和查找内容，以及通过按钮控制内容的音量。播放器在 iOS 平台上支持以下两种 AirPlay 模式。

- AirPlay Mirroring：允许将 iOS 设备屏幕输出到外部显示器。如果检测到 iOS 设备上正在使用镜像，则播放器会在 iOS 设备屏幕和外部显示屏上显示内容；
- AirPlay Streaming Export：允许将内容从 iOS 设备重定向到 Apple TV。

iOS 上的播放器 AirPlay Streaming Export 具有以下限制：

- 操作系统（OS）：iOS
- 平台：iOS 8.x 或更高版本，iTunes 10.2 或更高版本
- 流协议：DASH，HLS 和平滑流的实时和 VOD
- 视频解码器：H.264
- 音频解码器：AAC
- 播放器 Airplay 支持加密的内容播放

2.2.9 Android 平台上的 Tunneled 视频播放

Tunneled 视频播放有望带来好处，例如更好的音频/视频同步（AV 同步）和更流畅的播放。请参阅以下方法：

```
1. /**
2. * Enable/Disable Tunneling.
3. * @param value [in] Enable/Disable; true to enable, false to
   disable.
4. * @return {@link VO_OSMP_RETURN_CODE#VO_OSMP_ERR_NONE} if
   successful
5. */
6. VO_OSMP_RETURN_CODE enableTunneling(boolean value);
7. VOCommonPlayerConfiguration
```

2.2.10 Trick 模式

带有 I 帧的 Trick 模式支持（快进/快退），用于 DASH，符合 DASH-IF-IOP 建议。可以通过启用 **Special Feature --> Play I Frame Only** 的示例播放器来启用此功能。

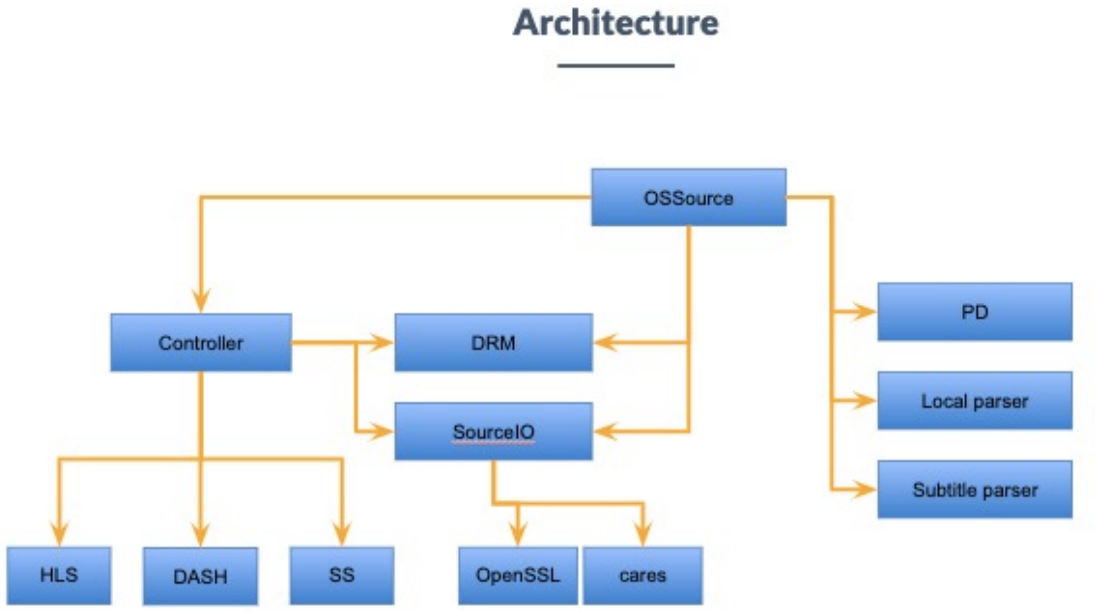
有关 API 的详细说明，例如：

```
1. (VO_OSMP_RETURN_CODE) setPlayIFrameOnly:(BOOL)enable
   Speed:(float)speed.
```

2.3 播放器中间层功能

播放器的中间层主要负责上层和底层的沟通，传递上层设置的各

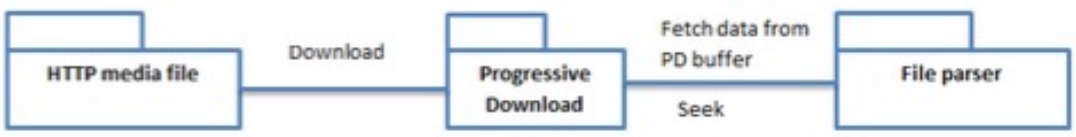
种 API 接口参数，并把这些数据解析，拆分，传递，返回等工作。主要交互模块有 OSSource，Controller，SourceIO，OSplayer 等。主要构架如下图所示，并在本章节介绍中间层的主要功能。



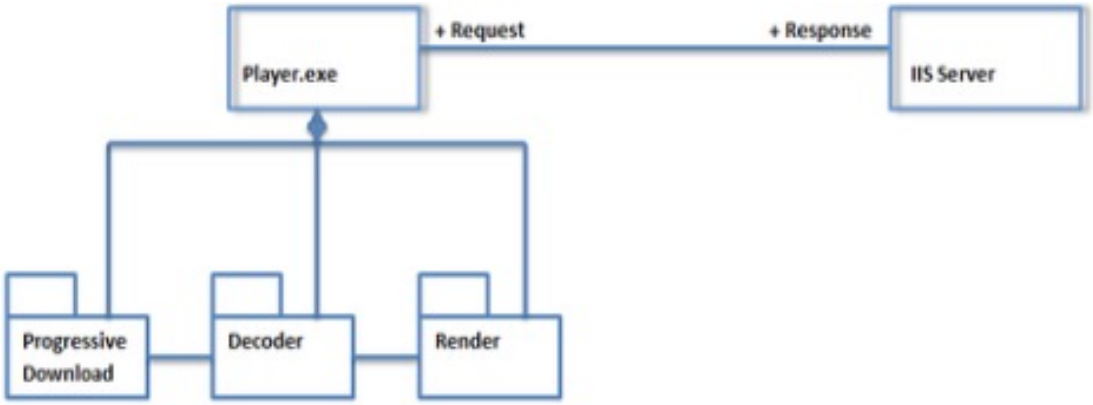
2.3.1 流媒体协议

2.3.1.1 渐进式下载协议 Progress Download

渐进式下载是从服务器到客户端的数字媒体文件传输，通常是在从计算机启动时使用 HTTP 协议进行传输终。终端用户可以在下载完成之前开始播放媒体。改播放器将媒体数据从 http 服务器下载到本地缓冲区（例如本地内存或文件），有效地管理有限的缓冲区，然后根据需要向文件解析器提供所需的数据。



要运行渐进式下载，用户需要将至少 40 MB 的正在播放的媒体文件下载到设备的内存中。如果所需的媒体数据不在缓冲区中，则播放器从 HTTP 服务器下载数据。如果所需的媒体数据在缓冲区中，则播放器从本地缓冲区读取并返回数据。



2.3.1.2 实时流协议 RTSP

实时流协议（RTSP）是一种网络控制协议，旨在用于娱乐和通信系统中以控制流媒体服务器。该协议用于在端点之间建立和控制媒体会话。媒体服务器的客户端发出 VCR 样式的命令（例如播放和暂停），以便于实时控制服务器中媒体文件的播放。流数据本身的传输不是 RTSP 协议的任务。大多数 RTSP 服务器使用实时传输协议（RTP）和实时控制协议（RTCP）进行媒体流传递。但是，某些供应商实现专有的传输协议。 例如，RealNetworks 的 RTSP 服务器软件还使用了 RealNetworks 专有的 Real Data Transport（RDT）。

播放器支持 RTSP 中如下的音频和视频解码：

1. Audio

- MPEG4-GENERIC
- MPEG4-LATM
- AMR-WB+
- AMR-WB
- AMR-NB
- QCELP
- WMA
- Video
- H.264
- MPEG4
- H.263
- WMV

播放器中的 RTSP 支持下列相关协议：

1. RFC 2326: Real Time Streaming Protocol (RTSP)
2. RFC 2327: SDP: Session Description Protocol
3. RFC 3550: RTP: A Transport Protocol for Real-Time Applications

下面是支持的 RTP 协属性：

1. RFC 3551: RTP Profile for Audio and Video Conferences with Minimal Control

以下是支持的 RTP 有效负载类型：

1. RFC 3984: RTP Payload Format for H.264 Video
2. RFC 3016: RTP Payload Format for MPEG-4 Audio/Visual Streams

3. RFC 4629: RTP Payload Format for ITU-T Rec.H.263 Video
4. RFC 3267: Real-Time Transport Protocol (RTP) Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (ARM-WB) Audio Codecs
5. RFC 4352: RTP Payload Format for the Extended Adaptive Multi-Rate Wideband (AMR-WB+) Audio Codec
6. RFC 2658: RTP Payload Format for PureVoice (tm) Audio
7. MS-RTSP: RTP Payload Format for ASF Streams

2.3.1.3 Windows Media HTTP

Windows Media HTTP（WMHTTP）流协议是基于客户端/服务器的协议，用于在客户端（流数据的接收方）和服务器（流数据的发送方）之间流实时数据。播放器支持以下 WMHTTP 功能：

1. 支持非管道模式和管道模式
   - 非管道模式：一种操作模式，其中来自客户端的请求必须通过与服务器用于将内容流式传输到客户端的 TCP 连接分开的 TCP 连接发送；
   - 管道模式：一种操作模式，在该模式下，可以在服务器用于将内容流式传输到客户端的同一 TCP 连接上发送来自客户端的请求。
2. 支持单文件流和实时流
   - 单文件流传输：有关更多信息，请参见单文件流传输；

- 实时流式传输：在仍由编码器编码时流式传输的内容。

3. 指定传输速率

- 指定多媒体数据量（以毫秒为单位），客户端请求服务器加快速度；
- 保持在线请求可防止服务器超时；
- 将有关流内容的统计信息提交到服务器。

2.3.1.4 MPEG-OMAF

Omnidirectional MediA Format (OMAF) 全向媒体格式是由 Moving Picture Expert Group (MPEG) 运动图像专家组所打造的 360 度媒体内容标准。OMAF 为全向内容具定义了 3 个级别的自由度 (3DOF)，例如 360° 视频，图像，音频和定时文本。并且将与视口无关的 360° 视频转换为符合 OMAF 的内容仅需要文件格式和传输协议级别的修改（例如，基于 MP4 和 DASH 的分段流）。

OMAF HEVC“视口相关”的基础提高了视频编码的要求，因为它用不同的质量和分辨率对前景（视口）和背景进行编码。视口自适应操作通过将重点集中在用户正在观看的区域上，减少 360° 视频所需的带宽。由于用户可以自由转动头部并更改活动视口，因此要求系统必须能够快速做出反应，以把不同区域的视频切换到高质量。

从解码和播放渲染客户端来说，OMAF Player 只需要知道 DASH

清单的 URL 就可以开始播放。从本质而言还是一个 DASH 的播放器，OMAF 将与 VR 相关的元数据添加到 ISOBMFF 和 DASH 清单中，播放器根据当前 IMU 融合的空间姿态四元数与 OMAF 将与 VR 相关的元数据比对获取相应质量的高清播放流。

2.3.2 自适应流传输模式

自适应流传输是一种用于通过计算机网络流传输多媒体的技术，该技术通过实时检测用户的带宽和 CPU 容量并相应地调整视频流的质量来工作。它需要使用能够以多个比特率编码单个源视频的编码器。 播放器会根据可用资源在流式传输不同编码之间进行切换。自适应流包括以下三种类型：

1. HTTP 实时流（HLS）
2. Microsoft 平滑流
3. HTTP 上的动态自适应流（DASH）

2.3.2.1 HLS —— HTTP Live Streaming

HTTP Live Streaming（缩写是 HLS）是由苹果公司提出基于 HTTP 的流媒体网络传输协议。是苹果公司 QuickTime X 和 iPhone 软件系统的一部分。它的工作原理是把整个流分成一个个小的基于 HTTP 的文件来下载，每次只下载一些。当媒体流正在播放时，客户端可以选择从许多不同的备用源中以不同的速率下载同样的资源，允许流媒体会话适应不同的数据速率。在开始一个流媒体会话

时，客户端会下载一个包含元数据的 extended M3U (m3u8) playlist 文件，用于寻找可用的媒体流。

HLS 只请求基本的 HTTP 报文，与实时传输协议（RTP）不同，HLS 可以穿过任何允许 HTTP 数据通过的防火墙或者代理服务器。它也很容易使用内容分发网络来传输媒体流。

相对于常见的流媒体直播协议，例如 RTMP 协议、RTSP 协议、MMS 协议等，HLS 直播最大的不同在于，直播客户端获取到的，并不是一个完整的数据流。HLS 协议在服务器端将直播数据流存储为连续的、很短时长的媒体文件（MPEG-TS 格式），而客户端则不断的下载并播放这些小文件，因为服务器端总是会将最新的直播数据生成新的小文件，这样客户端只要不停的按顺序播放从服务器获取到的文件，就实现了直播。由此可见，基本上可以认为，HLS 是以点播的技术方式来实现直播。由于数据通过 HTTP 协议传输，所以完全不用考虑防火墙或者代理的问题，而且分段文件的时长很短，客户端可以很快的选择和切换码率，以适应不同带宽条件下的播放。不过 HLS 的这种技术特点，决定了它的延迟一般总是会高于普通的流媒体直播协议。

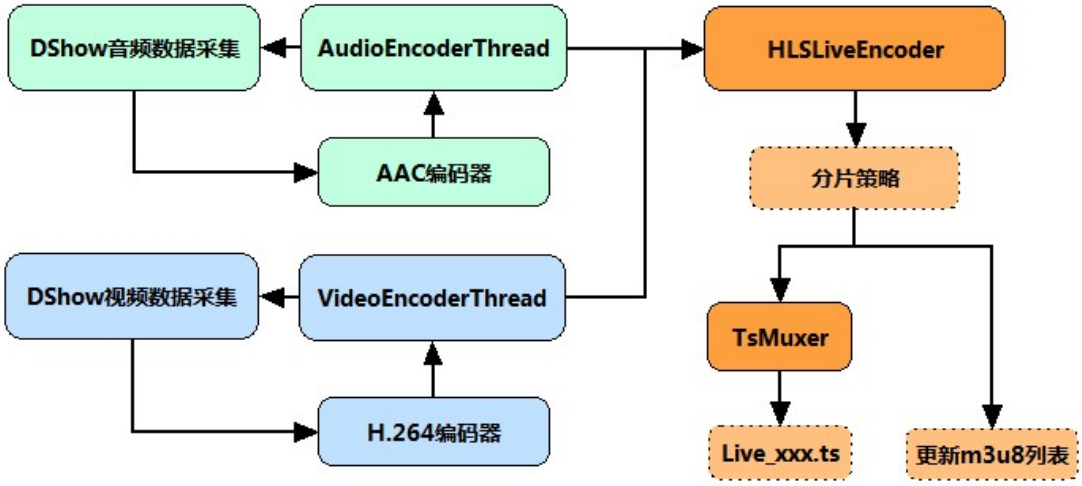
根据以上的了解要实现 HTTP Live Streaming 直播，需要研究并实现以下技术关键点：

1. 采集视频源和音频源的数据
2. 对原始数据进行 H264 编码和 AAC 编码
3. 视频和音频数据封装为 MPEG-TS 包

4. HLS 分段生成策略及 m3u8 索引文件

5. HTTP 传输协议

通过以上分析，实现 HLS LiveEncoder 直播编码器，其逻辑和流程基本上很清楚了：分别开启音频与视频编码线程，通过 DirectShow（或其他）技术来实现音视频采集，随后分别调用 libx264 和 libfaac 进行视频和音频编码。两个编码线程实时编码音视频数据后，根据自定义的分片策略，存储在某个 MPEG-TS 格式分段文件中，当完成一个分段文件的存储后，更新 m3u8 索引文件。如下图所示：



上图中 HLSLiveEncoder 当收到视频和音频数据后，需要首先判断，当前分片是否应该结束，并创建新分片，以延续 TS 分片的不断生成。需要注意的是，新的分片，应当从关键帧开始，防止播放器解码失败。核心代码如下所示：

```
        // 从关键帧开始
        if (isKeyframe)
        {
            // 检查是否应当创建新的分片文件
            if (CheckRecordTimestamp(timestamp))
            {
                if (ts_muxer_)
                {
                    std::string tsfilename = ts_muxer_->Filename();
                    ts_muxer_->WriteEnd();
                    delete ts_muxer_;
                    ts_muxer_ = 0;

                    // 更新m3u8索引文件
                    UpdateM3U8List(tsfilename, real_duration_/1000);
                }

                // 生成新的分片文件名
                std::string newfilename = GetRecordFilename();
                ts_muxer_ = new TSMuxer(newfilename.c_str());
                ts_muxer_->WriterHeader();
            }
        }
```

TsMuxer 的接口也是比较简单的。

```
#include "mpegts/mpegtsenc.h"

#define _USE_TS_MUXER_LOG_

class TSMuxer
{
public:
    explicit TSMuxer(const char* filename);

    ~TSMuxer();

    std::string Filename() { return filename_; }

    // 初始化，创建MPEG-TS音频和视频输出流
    void WriterHeader();

    // 将内存中剩余数据写入文件
    void WriteEnd();

    // 写入视频数据
    void WriteVideoData(char* buf, int bufLen, bool isKeyframe, unsigned int timestamp);

    // 写入音频数据
    void WriteAudioData(char* buf, int bufLen, unsigned int timestamp);

    // 废弃
    void WriteVideoSeqheader(char* buf, int bufLen);

private:
    std::string filename_;
    __int64 time_begin_;
    FILE* fp_;
    bool is_video_begin_;
    MpegTSWrite* mpegts_write_;
```

苹果公司把 HLS 协议作为一个互联网草案（逐步提交），在第一阶段中已作为一个非正式的标准提交到 IETF。2017 年 8 月，RFC 8216 发布，描述了 HLS 协议第 7 版的定义。

以下是播放器支持的标签。

- EXT-X-BYTERANGE
- EXT-X-DATERANGE
- EXT-X-DEFINE
- EXT-X-DISCONTINUITY
- EXT-X-ENDLIST
- EXT-X-I-FRAME-STREAM-INF
- EXT-X-I-FRAMES-ONLY
- EXT-X-INDEPENDENT-SEGMENTS
- EXT-X-KEY(IV)
- EXT-X-KEY(URI)
- EXT-X-MAP
- EXT-X-MEDIA
- EXT-X-MEDIA(SUBTITLED/FORCED/CHARACTERISTICS)
- EXT-X-MEDIA-SEQUENCE
- EXT-X-PLAYLIST-TYPE
- EXT-X-PROGRAM-DATA-TIME
- EXT-X-SESSION-DATA
- EXT-X-SESSION-KEY
- EXT-X-START
- EXT-X-STREAM-INF
- EXT-X-STREAM-INF(AUDIO/VIDEO)

- EXT-X-STREAM-INF(SUBTITLES)
- EXT-X-TARGETDURATION
- EXT-X-VERSION

播放器支持以下在 HLS 里的媒体格式

- MPEG2-TS
- MPEG-ES/AAC
- MPEG-ES/MP3
- ID3(MPEG-ES)
- ID3(MPEG2-TS)
- WebVTT

2.3.2.2 DASH —— HTTP 动态自适应流

HTTP 动态自适应流（DASH），也称为 MPEG-DASH，是一种自适应比特率流技术，可通过 Internet 从常规 HTTP Web 服务器提供高质量的媒体内容流。与 Apple 的 HTTP Live Streaming（HLS）解决方案类似，MPEG-DASH 的工作原理是将内容分成一系列小的基于 HTTP 的文件段，每个段包含一段短时间的内容回放时间，这可能会持续数小时，例如电影或体育赛事的直播。

当内容由 MPEG-DASH 客户端播放时，客户端会根据当前的网络状况自动从替代项中选择下一个要下载和播放的片段。客户端选择可能具有最高比特率的段，可以及时下载该段进行回放，而不会引起停顿或重新缓冲事件。因此，MPEG-DASH 客户端可以无缝地

适应不断变化的网络状况，并提供高质量的播放而不会出现停顿或重新缓冲事件的情况。

DASH 是由 MPEG (Moving Picture Experts Group)组织制定，2010 年开始启动，2011 年 11 月发布 Draft 版本，2012 年 4 月发布第一稿 Version（ISO/IEC 23009-1:2012），2014 年 5 月发布第二稿（ISO/IEC 23009-1:2014），最新稿（ISO/IEC 23009-3:2015）。

目前 3GPP Release 10 已经将 DASH 纳入其中；在 HbbTV 1.5 中也支持 DASH；DVB-DASH 也将 DASH 纳入到 DVB（ETSI TS 103 285 v.1.1.1）。目前 DASH Industry Forum 由发起厂家组成，致力于推进 DASH 产品生态，将 DASH 产业化和业界最佳实践推向批量应用。

ISO/IEC 23009 包含四个部分:

- Part 1: Media presentation description and segment formats
- Part 2: Conformance and reference software
- Part 3: Implementation guidelines
- Part 4: Segment encryption and authentication

DASH 相比 HLS、HSS 等协议的优势在于:

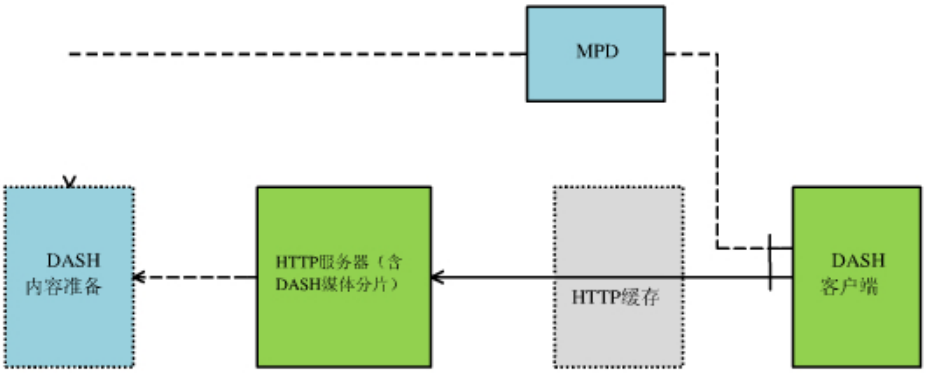
1) DASH 支持多种编码，支持 H.265、H.264、VP9 等等；
2) DASH 支持 MultiDRM，支持 PlayReady、Widewine，采用通用加密技术，支持终端自带 DRM，可以大幅度降低 DRM 投资成本；
3) DASH 支持多种文件封装，支持 MPEG-4、MPEG-2 TS

（Transport Stream）；

4) DASH 支持多种 CDN 对接，采用相同的封装描述对接多厂家 CDN；
5) DASH 支持异构终端，浏览器原生不用插件就可以支持，Android/iOS/Windows/Flash 可以通过 JITP 将 DASH 转换为 HLS、HDS、HSS 等，已支持 Legacy 终端类型，支持一份存储，大幅度减少文件存储量；
6) DASH 支持直播、点播、录制、时移等等丰富的视频特性；
7) DASH 支持动态码率适配，支持多码率平滑切换；
8) DASH 支持紧缩型描述以支持快速启动；
9) DASH 支持客户端和服务端的广告插入。

DASH 典型的一个端到端的系统包含 Encoder、Dash Server、Origin Server、Edge Server、DASH Client：

Media Presentation Description 是描述分片和时序的信息：从 MPD->Period->Adpatation Set(Video/Audio Track)->Representation(Multiple bitrate)->Segement



DASH 内容准备提供不同码率的视频文件（使用不同的质量需求和网络环境），然后使用特定的分片方法， 将视频文件分片，然后分片传输到客户端进行播放。MPD 文件时在视频文件进行分片时获得的视频本身的属性信息和视频分片信息。

DASH 支持以下视频和音频编解码器:

1. Video codec

- H.264
- WMV/VC1
- H.265

2. Audio codec

- AAC
- WMA

DASH 支持的功能包括:

- Request and response through HTTP
- XML parser used for Media Presentation Description (MPD) file
- Audio and video fragment process mechanism
- MP4 fragment parser callback
- TS fragment parser callback
- Playback of multi-period VoD content
- Playback of indexed content
- Playback of PD content (one file)
- Playback of live content

- Playback of VoD TTML or SMPTE subtitle

DASH 使用 ISO，IEC 和 23009 标签。媒体表示描述（MPD）文档包含 DASH 客户端所需的元数据，以构造适当的 HTTP URL 来访问段并向用户提供流服务。

Media Presentation 包含一个或多个 Periods。Multi_Period 用于广告插入。

- AdaptationSet：每个 Period 包含一个或多个 AdaptationSet。通常，AdaptionSet 分为三种类型的流：视频，音频和字；
- ContentComponent：每个 AdaptationSet 包含一个或多个媒体 ContentComponents。每个媒体 ContentComponent 的属性由 ContentComponent 元素描述，或者如果 Adaptation Set 中仅存在一个媒体 ContentComponent，则可以直接在 AdaptationSet 元素上描述。

一个 Presentation 是在定义的期间内包含媒体内容的媒体内容组件的完整集合或子集的替代选择之一。

- Segment 由 SegmentTemplate 元素定义。在这种情况下，特定的标识符将由分配给细分的动态值代替，以创建细分列表；
- SegmentTimeline 元素表示表示形式中每个段的最早表示时间和表示持续时间（以基于@timescale 属性为单位）。

2.3.2.3 比特率自适应

比特率自适应是一种在各种环境条件下有效传递流内容以获得最

佳播放质量的过程。播放器支持基于 HTTP 的自适应流协议（即 HLS，平滑流和 DASH）的比特率自适应。播放器考虑了网络拥塞，设备功能，设备负载以及其他几个因素，以确定要播放的最佳视频比特率。自适应功能可在不同视频质量之间无缝过渡，而不会中断播放。播放器遵循以下规则来执行比特率适配操作：

- HLS: 播放器选择默认的视频/音频轨道；
- SS/DASH: 播放器选择清单文件中定义的第一个视频/音频轨道。

除了自动比特率自适应以外，OSMP +还支持许多配置选项以完善自适应行为。这些配置选项包括：

- 初始（开始）比特率
- 开始缓冲时间
- 播放重新缓冲时间
- 最大缓冲时间
- 最小比特率
- 最大比特率
- 基于 CPU 的适应
- 设备功能配置

将这些选项与自动比特率自适应技术相结合，可提供一种灵活的解决方案，以满足各种需求。

2.3.3 多音轨

在自适应流传输中，如果在 Manifeat 文件中包含了多个音轨，则播放器会提取信息并提供一种机制来选择要播放的特定音轨。播放器还提供了 API，以查询与音频有关的许多属性，例如可用替代音频的数目，编解码器类型，采样率，通道数，语言等。

播放器支持在播放过程中动态更改音轨的功能。在可以使用多种语言的音频轨道的情况下，为了简化针对不同地理区域的内容准备，可以指定首选语言，以便从最有利的语言开始播放。

2.3.4 字幕

字幕显示对用户来说是至关重要，许多观众利用字幕来代替耳机，特别是在欧洲（多语言）和在美国（CC608/708 是必须支持的字幕)。国内的用户观看国外影片时也多需要字幕作为辅助。本项目播放器能够支持多种形式的字幕扩展功能，包括 WEBVTT，SMPTE，SRT，SMI 等外部字幕形式和 ISMT，DVB，W3C 等内部字幕形式，支持 close caption 608/708。

同时，为了更好的支持业务的开展，播放器还可以支持多路音轨和字幕的播放及切换，方便根据不同的需求来选择合适的字幕。

2.3.4.1 隐藏式字幕

播放器支持以下隐藏式字幕格式（包括两种主要样式：绘画/滚动和弹出/弹出)：

- CEA-608
- CEA-708

CEA608 开发于 1970 年代，用于在 NTSC 视频（模拟复合视频格式）的 VBI（垂直消隐间隔）中承载 960bps 字幕数据和服务。在后来的 NTSC 数字分量版本中，该模拟波形被数字编码并嵌入到视频的第 21 行中。

CEA-708 定义了 DTV 隐藏式字幕（DTVCC），并为字幕服务提供商，电视信号的发行商，解码器和编码器制造商，DTV 接收器制造商以及 DTV 信号处理设备制造商提供了规格和指南。

播放器支持以下隐藏式字幕功能：

- Text
- Horizontal Justification (Left, Right, and Center)
- Vertical Justification (Top, Bottom, and Center)
- Print Direction (Left-Right, Right-Left, Top-Bottom, and Bottom-Top)
- Text Wrap
- Scroll Direction (Left-Right, Right-Left, Top-Bottom, and Bottom-Top)
- Rectangle Border (None, Raised, Depressed, Uniform, Shadow Left, and Shadow Right)
- Rectangle Border Color
- Rectangle Border Fill Color

- Rectangle Display Effect (Snap, Fade, and Wipe)
- Rectangle Display Effect Direction (LR, RL, TB, and BT)
- Rectangle Display Effect Speed
- Font Size (Standard, Small, and Large)
- Font Color (RGBA, 32bit)

播放器支持以下隐藏式字幕字体样式：

- Default (undefined)
- Monospaced with serifs
- Proportionally spaced with serifs
- Monospaced without serifs
- Proportionally spaced without serifs
- Casual font type
- Cursive font type
- Small capitals

播放器支持以下隐藏式字幕文本标签：

- Dialog
- Source_speaker_ID
- Electronically_reproduced_voice
- Dialog_language_other_than_primary
- Dialog_Voiceover
- Dialog_Audible_Translation
- Dialog_Subtitle_Translation

- Dialog_Voice_quality_description
- Dialog_Song_Lyrics
- Dialog_Sound_effect_description
- Dialog_Musical_score_description
- Dialog_Expletive
- Dialog_Text_not_to_be_displayed

播放器为应用程序提供 API，以控制隐藏式字幕文本的属性，包括：

- Text size
- Text color and opacity
- Text effects, for example, italic, underline, bold
- Background color and opacity
- Window color and opacity
- Font selection

2.3.4.2 其他字幕

播放器还支持以下字幕格式：

- TTML
- SAMI
- Envivio
- SMPTE
- SRT

- DVB (exclude teletext)
- WEBVTT

播放器支持以下编码格式：

- EUC-KR
- windows-949
- GB2312
- UTF-8
- ISO-8859-1
- ISO-8859-2
- ISO-8859-3
- ISO-8859-4
- ISO-8859-5
- ISO-8859-6 (Android and iOS)
- ISO-8859-7 (Android and iOS)
- ISO-8859-8 (Android and iOS)
- ISO-8859-9
- ISO-8859-10 (Android, Windows, and Mac OS)
- ISO-8859-11 (Android)
- ISO-8859-13
- ISO-8859-14 (Windows and Mac OS)
- ISO-8859-15
- ISO-8859-16 (Windows and Mac OS)

- Shift_JIS
- Unicode

播放器支持以下字幕功能：

- Text
- Horizontal Justification (Left, Right, and Center)
- Vertical Justification (Top, Bottom, and Center)
- Print Direction (Left-Right, Right-Left, Top-Bottom, and Bottom-Top)
- Text Wrap
- Scroll Direction (Left-Right, Right-Left, Top-Bottom, and Bottom-Top)
- Rectangle Border (None, Raised, Depressed, Uniform, Shadow_Left, and Shadow_Right)
- Rectangle Border Color
- Rectangle Border Fill Color
- Rectangle Display Effect (Snap, Fade, and Wipe)
- Rectangle Display Effect Direction (LR, RL, TB, and BT)
- Rectangle Display Effect Speed
- Font Size (Standard, Small, and Large)
- Font Color (RGBA, 32bit)

播放器支持以下字幕字体样式：

- Default (undefined)

- Monospaced with serifs
- Proportionally spaced with serifs
- Monospaced without serifs
- Proportionally spaced without serifs
- Casual font type
- Cursive font type
- Small capitals

播放器为应用程序提供 API，以控制字幕文本的属性，包括：

- Text size
- Text color and opacity
- Text effects, for example, italic, underline, bold
- Background color and opacity
- Window color and opacity
- Font selection

2.3.4.3 符合 FCC 规范的特性

(一) 演示

所有设备都必须呈现字幕，以便字幕文本可以出现在一个或单独的字幕窗口中，并支持以下模式：

- 一次全部显示的文本（pop-up）；
- 随着新文本出现而向上滚动的文本（rollup）；
- 每个新字母或单词到达时出现的文本（paint-on）。

播放器没有支持演示模式的 API。相反，OSMP +会根据 CEA-608 和 CEA-708 规范从流中获取此数据。

(二) 字符大小

所有设备都必须实现字幕，以便用户可以更改字幕文本的大小，并且需要提供从默认字符大小的 50％到默认字符大小的 200％的大小范围。要设置 FCC 字符大小，请使用 setSubtitleFontSizeScale API。

(三) 字符颜色

所有设备都必须加字幕，以便字符可以显示为 CEA－708 定义的 64 种颜色，并且用户可以覆盖字符的创作颜色，并可以从至少 8 种颜色的调色板中进行选择，包括白色，黑色，红色，绿色，蓝色，黄色，洋红色和青色。要设置 FCC 字符颜色，请使用 setSubtitleFontColor API。

(四) 字符不透明度

所有设备都必须实现字幕，以便为用户提供更改字幕文本的不透明度以及在不透明和半透明不透明度之间进行选择的功能。要设置 FCC 字符的不透明度，请使用 setSubtitleFontOpacityAPI。

(五) 字体

所有设备都必须实现字幕，以便字体可用于实现 CEA－708 要求的八种字体。必须为用户提供将设备中包含的字体分配为 Android 和 iOS 上以下八种样式的默认字体：

- Default (undefined)

- Monospaced with serifs (similar to Courier)
- Proportionally spaced with serifs (similar to Times New Roman)
- Monospaced without serifs (similar to Helvetica Monospaced)
- Proportionally spaced without serifs (similar to Arial and Swiss)
- Casual font type (similar to Dom and Impress)
- Cursive font type (similar to Coronet and Marigold)
- Small capitals (similar to Engravers Gothic)

(六) 字幕背景颜色和不透明度

所有设备都必须实现字幕，以便字幕背景可以显示为 CEA－708 定义的 64 种颜色，并且用户可以覆盖字幕背景的创作颜色并从至少 8 种颜色的调色板中进行选择，包括白色，黑色，红色，绿色，蓝色，黄色，洋红色和青色。

要设置 FCC 字幕背景，请使用 setSubtitleFontBackgroundColorAPI。

所有设备都必须实现字幕，以便为用户提供更改字幕背景的不透明度并在不透明，半透明和透明背景不透明度中进行选择的能力。

要设置 FCC 字幕的不透明度，请使用 setSubtitleFontBackgroundOpacity API。

(七) 字符边缘属性

所有设备都必须实现字幕，以便出现字符边缘属性，并且为用户提供了选择包括以下内容的字符边缘属性的能力：

- 没有边缘属性

- 凸起的边缘
- 凹陷的边缘
- 均匀的边缘
- 阴影阴影边缘：左阴影/右阴影

要设置 FCC 字符边缘属性，请使用 setSubtitleFontEdgeType API。

(八) 字幕窗口的颜色和不透明度

所有设备都必须实施字幕，以便字幕窗口颜色可以显示为 CEA－708 定义的 64 种颜色，并且用户可以覆盖字幕窗口的创作颜色并从至少 8 种颜色的调色板中进行选择，包括白色，黑色，红色，绿色，蓝色，黄色，洋红色和青色。

要设置 FCC 字幕窗口的颜色，请使用 setSubtitleWindowBackgroundColorAPI。

所有设备都必须实现字幕，以便为用户提供更改字幕窗口的不透明度并在不透明，半透明和透明背景不透明度中进行选择的能力。

要设置 FCC 字幕窗口的不透明度，请使用 setSubtitleWindowBackgroundOpacity API。

(九) 语言

当呈现这样的字幕时，所有设备都必须具有在其他语言的字幕轨道之间进行选择的能力，并且提供了当此类字幕可用时用户选择简化或缩小字幕的能力，并将这种字幕轨道标识为“easy reader”。

要设置 FCC 语言，请使用 setSubtitlePath API。

(十) 预览和设置保留

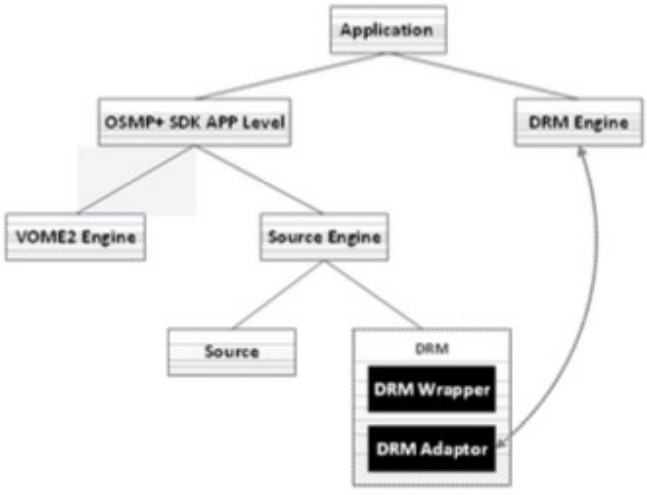
所有设备都必须为用户提供预览本节要求的字幕功能的默认值和用户选择的能力，并且必须保留默认字幕配置等设置，直到用户更改字幕配置为止。

要设置 FCC 预览和保留，请使用 PreviewSubtitle API。

2.3.5 数字加密模块 DRM

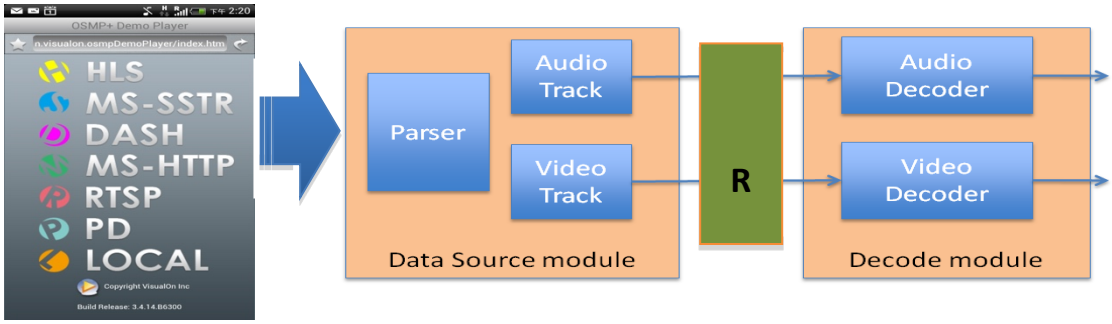
数字版权管理（DRM）是一类技术，硬件制造商，发行商，版权持有者和个人使用这些技术来控制售后数字内容（视频，音频等）和设备的使用。

播放器不依赖与任何 DRM 技术，并且可以与所有商业和专有 DRM 引擎集成，以在设备上播放受保护的内容。DRM 组件允许或阻止设备上数字资产的回放，而播放器处理与这些资产的高质量回放有关的步骤。DRM 引擎位于应用程序的下方，DRM Adapter 用于将私有 DRM 接口转换为通用接口供其他引擎使用。



如果视频应用只是通过集成 Android 系统默认播放器来提供视频

播放功能，那该应用只能被动使用系统自带的 DRM（数字加密模块)。而通过以下模组化的框架，在 DRM（数字加密模块）方面却具有独特的灵活性，可以根据需求来集成不同的 DRM 模块，可以根据不同项目的服务需求来选用对应的 DRM（数字加密模块)。建议采用上述类似框架完成 DRM（数字加密模块） 模块的集成。



2.3.6 IPV6-only 网络支持

仅 IPv6 网络支持在 iOS 和 Mac OS 上可用。

除了支持 IPv4 或双栈（IPv4 / IPv6）网络外，Apple 还要求为提交给 App Store 的所有应用程序支持仅 IPv6 网络。

播放器与从此版本开始的 DNS64 / NAT64 转换机制兼容，以便在客户端仅具有 IPv6 连接并尝试寻址 IPv4 上托管的内容时容纳工作流程。

如果用户尝试从仅 IPv6 的网络访问托管在 IPv4 上的内容，它将请求 DNS64 从 URL 域名获取 IP 地址，并在 IPv6 空间中接收表示 IPv4 地址的合成 IPv6 地址。使用此综合地址，用户可以使用 NAT64 建立到 IPv4 主机的连接，该连接将 IPv6 网络连接到 IPv4 网络。

另一方面，如果用户尝试从纯 IPv6 网络访问托管在 IPv6 上的内容，则它使用 DNS64 获取 URL 域的 IPV6 地址，并使用标准 IPv6 路由机制建立与内容的 IPv6 连接。

在下列平台上，此新功能可用于清除和加密内容：

- iOS 9.0 + / OS X 10.11.4+：同时支持 IPv4 内容的 URL 主机地址和 IP 文字地址；
- 低于 iOS 9.0 或 OS X 10.11.4 的版本：仅支持 IPv4 内容的 URL 主机地址。

以下是日志消息示例，可帮助您识别/验证此 DNS64 / NAT64 转换。

- 如果 IPv4 内容是 URL 主机地址，则您可以使用以下格式获取日志：

```
1. '15:08:38.319 @@@VOLOG, Info, ModuleID [xxxxxxxx], ThreadID
   [xxxxxxxx], vo_http_stream.cpp, send_request, Line#1731,
   [SourceIO] address:64:ff9b::a02:4407, url:example.visualon.com'
```

- 如果 IPv4 内容是 IP 地址，则您可以使用以下格式获取日志：

```
1. 'vo_socket.cpp, VO_IOS_DNSResolve, Line#626, IP item: i=0,
   host=10.2.68.7, port=8082, is_ipv6=1, addr=64:ff9b::a02:4407,
   dataLen=28, sa_len=28, fam=30'.
```

2.3.7 视频水印

VideoMark 是基于会话的视频水印，用于法医跟踪。它旨在为整个生命周期内的高价值数字内容（例如超高清）或早期 VOD 发行提供保护。它旨在将水印嵌入客户端设备中，与服务器端水印相比，它被证明是更具扩展性的解决方案。

第三方开发的 VideoMark 技术也已与播放器完全集成。该解决方案用于在播放之前将唯一且难以察觉的标记嵌入视频流的帧中。它使用软和硬编解码器集成，并且可在 Android 和 iOS 设备上使用。

2.3.8 标准化 HTTP 302 重定向机制

在播放器默认情况下 HTTP 302 响应与 HTTP 302 规范（临时重定向）保持一致。并提供了一个选项，该选项允许还原到以前的重定向机制，该机制的 HTTP 302 被视为 HTTP 301（永久重定向）。这样做，请使用带有字符串{"http302redirect": "true"}的 addConfiguration（）API。

2.3.9 低延时和同步播放

播放器在 Android，iOS 和 HTML5 +平台上支持 DASH 实时流低延迟特性。有两种类型的 API，第一种类型用于启用低延迟模式；第二种类型用于设置目标等待时间值（同步回放）。请参考下表的低延迟 API 和表现。

API 类型	APIs 和 sourceConfig	行为

1. 打开低延时 API	enableLowLatencyMode	当 API 打开时，低延时模式打开，默认值为 3000 毫秒
2. 设置低延时的值	SetPresentationDelay	When the API is set, synchronization playback mode is on, and the target latency is the value set by this API 当 API 打开时，同步模式打开，然后用户可以修改此值。

API 的第二种类型是设置目标延迟值。(例如播放器上的 setPresentationDelay（） API)。该值指示在实际直播点之前设置了多少时间。通常，将目标等待时间设置为较低的值可以减少直播过程中的实际等待时间，但是，如果网络带宽不足，则将目标等待时间设置为低于 3000 毫秒可能会导致频繁加载的性能问题。 如果目标等待时间值设置为 0，则播放器将非常积极的赶上实时点。
延迟的目标持续时间是由 setPresentationDelay（）设置的值决定，该值指示在实际现场播放点之前设置了多少时间。默认情况下，setPresentationDelay（）的值为 3000 毫秒。

为了帮助在不同设备上同步播放 DASH 内容，可以使用以下机制来定义公共时间参考。如果 DASH 清单中有 UTCTiming 标签，它将用作同步的时间参考。 默认情况下启用。如果清单中有多个 UTCTiming 标签，则将考虑第一个 UTCTiming 标签，其他标签将被忽略。如果 DASH 清单中没有 UTCTiming 标签，则除非应用程序设置了另一个网络时间协议（NTP）时间参考，否则播放器将使用

http://time.akamai.com/?iso&ms 中的 UTC 时间。

要设置 UTC 时间，应用程序必须使用 addConfiguration 提供 NTP 时间方案和值。

例如，以下键值对可以作为 addConfiguration 的 JSON 字符串参数（确保将转义字符配置为适合目标 OS）：

```
1. {
2. "schemeIdUri": "urn:mpeg:dash:utc:http-head:2014", "value":
   "https://time.is/"
3. }
4. or
5. {
6. "schemeIdUri": "urn:mpeg:dash:utc:http-xsdate:2014",
7. "value": "https://time.is/"
8. }
```

以下是播放器中时间信息的说明示例。下图中的时间信息已在黄色进度条上从 T1 到 T6 标记，与上面的时间相对应。

- T1：启用“实时流 DVR 位置”时，该值始终为 DVR 窗口持续时间，表示最小实时位置；当禁用“实时流 DVR 位置”时，该值从 0 开始，并随着播放 的进行而平稳地增加；
- T2：该值是 UTC 时间与 UTC 播放时间之间的时间差。禁用“实时流 DVR 位置”时，该值从 0 开始并平稳增加；
- T3：UTC 播放位置；
- T4：UTC 时间；
- T5：PTS 位置（以秒为单位）；
- T6：启用“实时流 DVR 位置”时，该值始终为 0，表示最大实时位置，也表示实时位置；禁用“实时流 DVR 位置”时，该值从 DVR 窗口时间开始并持续增加。

3 产品兼容性及关键技术指标

3.1 产品功能的复杂性和优势

本方案所实现的系统充分考虑了灵活性和可升级性，采用通用的格式及协议定义各种接口，本系统方案不仅可以调用自有的服务模块，也具备调用第三方服务模块的功能。

本项目播放器采用完善的系统架构和专业的技术手段，实现了多种媒体超强的解码能力和业务适配能力，为运营商在视频应用领域上拓展提供了强有力的支撑。

系统具体有如下特点及优势：

1. **稳定性好**：播放器的稳定性严重影响用户的观感体验，是播放器要求的重中之重。本项目播放器具有极好的稳定性，在国内外的市场上大规模商业应用多年，拥有极好的业界口碑；
2. **兼容性强**：播放器需要适配多种系统平台和各式各样的硬件终端，所以对播放器的兼容性有很高的要求。本项目播放器能够很好的满足客户所要求的各类系统平台和硬件终端。
3. **可扩展性强**：本项目招标要求除了基本的播放功能外，还需要扩展广告插播、智能管控等其他定制化功能。因此对播放器的可扩展性要求高。本项目播放器通过强大的接口引擎，能够支撑各种业务功能的扩展，并已在长期的商用中积累了多种扩展功能。
4. **HEVC 软解性能强**：HEVC 是视频领域的趋势，必将被各厂家

所采用。本项目播放器在 HEVC 软解性能方面处于全球领先水平，已被国内外大型网络视屏运营商成功商用，获得用户好评。

5. **跨平台支持**: 本项目播放器能够支持 Android 与 iOS 等设备，也能通过 HTML5 的形式支持 PC 或者 MAC 上媒体播放。
6. **多终端多格式多字幕**: 硬件解码支持的设备多，流媒体协议支持全面 ，编码格式封装格式支持全，字幕扩展能力强。
7. **大数据分析功能**: 能够帮助运营商更好的了解内容的观看情况，对内容的组织运营有着非常重要的意义。本项目播放器具备大数据分析模块，此模块可以收集播放的各种数据（包括视频基本信息、网络状况、用户习惯、日志报告等）并发送到 Server。用户可以利用这些数据做出分析（包括网络分析，用户特点分析，错误报告分析等）以提高自身的产品质量和用户体验。
8. **完善的售后服务:** 能够及时提供技术支持和迅速解决突发问题的能力，支持客户的业务发展。

3.2 性能指标和优化

播放器不能能够提供丰富的功能和扩展性，在播放器性能上也做了深入的优化工作，具体数据如下:

首帧延时控低	直播(<1080p): 300ms 短视频(<1080p): 100~200ms 直播/点播(1080p): 700ms

	短视频(1080p): 100~200ms
seek 响应时长短	小于 1500ms
频道切换	直播小于 800ms；短视频小于 100~200ms
暂停	小于 20ms
继续播放	小于 30ms
切换字幕	直播小于 800ms；点播小于 700ms
音频切换	直播小于 1000ms；点播小于 800ms
DASH 现场直播延时	小于 3500ms

4 项目计划

4.1 项目开发计划表

序号	阶段名称	工作事项	时间	备注（实施前置条件）
1	需求阶段	设计架构确认	第 1 周-第 4 周	需要客户和供应商一起讨论整体构架设计确认。

		功能需求确认		需要客户方进行功能需求确认，只有功能需求确认后方可后续实施
		测试用例大纲确认		供应商按照客户的需求定义测试大纲，并让客户确认
		测试平台及测试流确认		需要客户确认测试平台硬件以及测试流媒体的覆盖性
2	设计阶段	演示播放器 demo 确认	第 5 周-第 15 周	基本功能确认工作，不包括特殊功能和性能优化
		播放器总体功能设计		播放器总体功能流程的确认工作
		播放器测试用例设计确认		供应商按照客户的需求设计完成测试用例，需要客户进行完整性确认

		API 接口设计确认		根据设计框架来定义 API 接口并需要客户确认
3	开发阶段	播放器功能开发	第 16 周-第 40 周	播放器开发阶段
		播放器接口开发		
4	测试阶段	一轮测试、bug 修改	第 41 周--第 50 周	项目整体测试阶段、分三轮进行
		bug 回归，二轮测试		
		bug 回归，三轮测试		
5	第一阶段试上线阶段	播放器部署上线测试（灰度测试）	第 51 周--第 52 周	播放器上线后进行灰度测试，排除主要问题后进行试运行
		播放器试运行		