WebRTC 的视频采集和视频流水线建立

视频采集模块在 WebRTC 数据流水线中负责从视频源采集视频数据，交给流水线下一模块(编码模块)进行处理。视频源除了常见的摄像头，也可以是桌面抓屏或者窗口抓屏，或者是一个视频文件。视频采集模块是平台相关的，MacOS 和 IOS 平台一般使用 AVFoundation 框架，Linux 平台一般使用 V4L2 库，而 Windows 平台则使用 MediaFoundation 库。

本文在深入分析 WebRTC 视频采集、SDP 协商和 VideoEngine 源代码的基础上，重点研究其视频采集模块的实现，视频数据流水线的建立过程，和视频数据从采集到待编码的流向，继续进一步学习 WebRTC 的音视频处理技术。

1 视频采集模块实现

如文章[1]所述，WebRTC 的视频采集模块处于数据流水线的开始位置，其实现采用 WebRTC 的通用模块机制。源代码分布在 webrtc/modules/video_capture 目录下，分为平台无关部分代码和平台相关部分代码：前者用来定义视频采集的通用接口如 StartCapture/StopCapture/RegisterCaptureDataCallback 等，后者则在不同平台上实现这些接口，如在 Mac 平台上使用 AVFoundation 实现开始/停止视频采集、视频数据向上推送等功能。

WebRTC 的视频采集模块 UML 类图如图 1 所示。

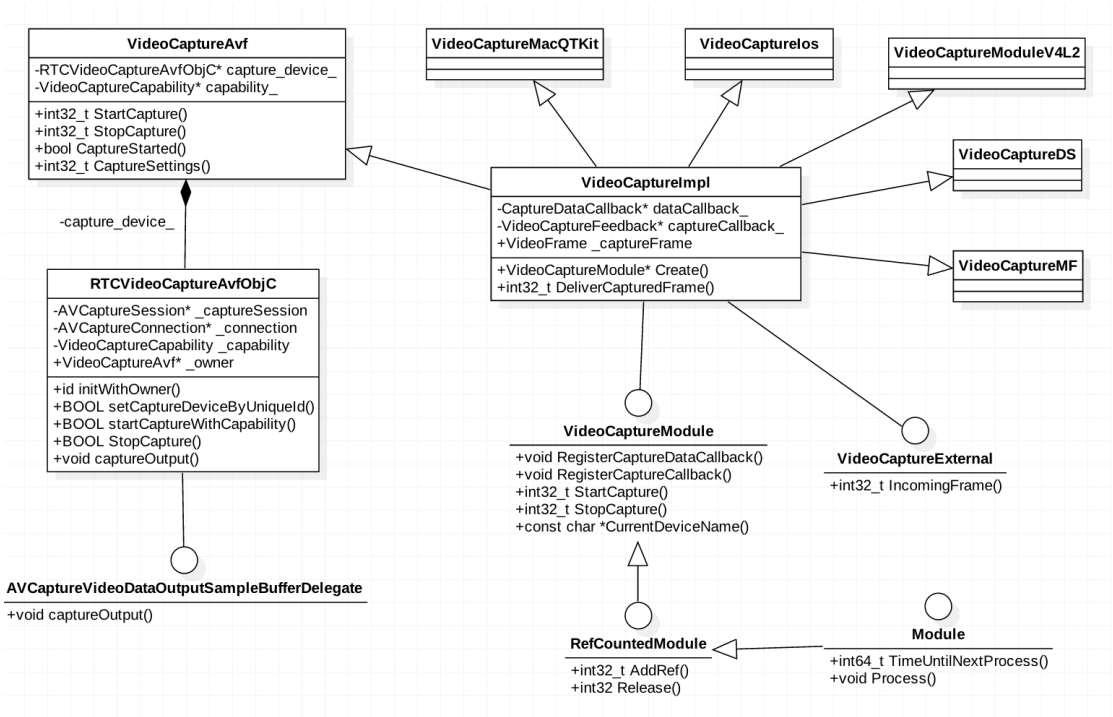

图 1 视频采集模块实现类图

视频采集模块的虚基类为 VideoCaptureModule，它定义一系列视频采集的通用接口函数：Start/StopCapture 用来开始/结束视频采集；Register/DegisterCaptureDataCallback 用来注册/注销数据回调模块，数据回调模块用来把视频数据向上层模块推送；CaptureCallback 则是向上层报告采集模块自身的运行状态。视频采集模块采用 WebRTC 的通用模块机制，因此它也继承自 Module 类，用来完成通用的模块操作。

VideoCaptureImpl 类是 VideoCaptureModule 的实现子类，它实现父类定义的通用平台无关接口。对于平台相关接口，则留在平台相关的子类中实现。该类定义一系列工厂方法来创建平台相关的具体子类。不同平台上实现的子类负责在各自平台下实现平台相关功能，主要是开始/结束视频采集和视频数据导出。在 Linux 平台上实现的子类是 VideoCaptureV4L2，在 IOS 平台的实现为 VideoCaptureIos，

在 Windows 平台上的实现为 VideoCaptureMF 和 VideoCaptureDS，等等。

本文以 Mac 平台的 VideoCaptureAvf 实现为例进行简要分析。VideoCaptureAvf 采用 AVFoundation 库实现视频采集功能，它把真正要做的事情委托给 RTCVideoCaptureAvfObjC 对象，后者同时实现了 AVCaptureVideoDataOutputSampleBufferDelegate 协议，采集到的视频数据通过该协议的 captureOutput 接口导出，由 IncomingFrame()函数向上传递到 VideoCaptureImpl 进行下一步处理。开始/结束视频采集由 AVCaptureSession 负责实现。

至此，我们分析了 WebRTC 中视频采集模块的实现细节。值得注意的是，在 Chrome52 Codebase 中，该模块并不支持 WindowsPhone 平台。另外，Android 平台的视频采集不通过该模块进行，其相关代码分布在 webrtc/api/java 目录下。

2 视频数据流水线建立

视频采集模块作为底层模块，需要和上层模块协作才能把采集到的视频数据发送到上层的显示和编码模块，为数据流水线提供源源不断的视频数据。从控制流来讲，视频采集模块在初始化阶段由上层模块进行创建并开启视频采集，在结束的时候由上层模块停止视频采集并销毁模块。从数据流来讲，采集到的视频数据通过回调接口传递到上层模块，进行数据流水线上的下一步处理。

本节分析和视频采集紧密相关的发送端数据流水线建立过程，主要包括视频采集

模块创建后经过 VideoSource、VideoTrack、MediaStream 向上添加到到 PeerConnection，SDP 本地音视频能力收集创建 offer，以及 offer 创建后的本地设置生效。下面详细描述之。

2.1 相关 UML 类图

WebRTC 视频数据流水线的相关代码分布在 webrtc/modules/video_capture，webrtc/api，webrtc/media 等目录中，这些共同协作完成视频数据流水线的建立。相关 UML 类图如图 2 和图 3 所示。

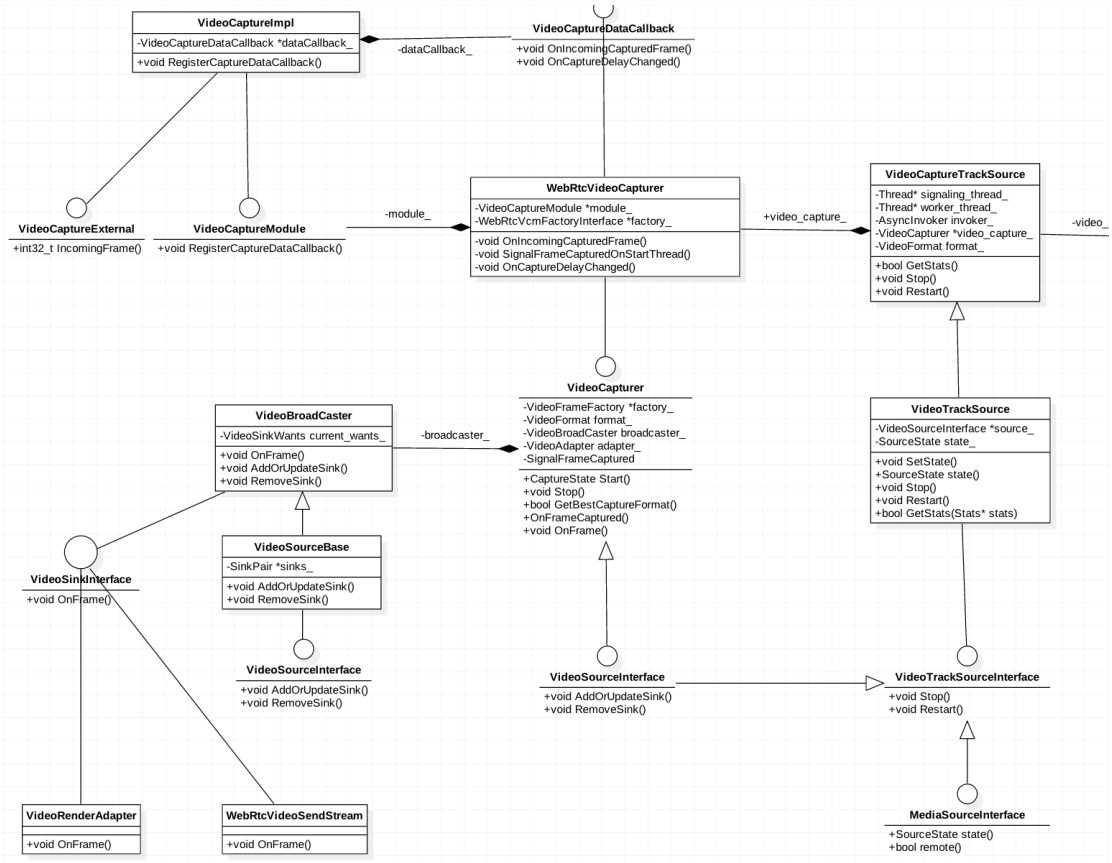

图 2 从 VideoCapture 到 VideoCaptureTrackSource

VideoCapture 模块对内完成视频采集任务，对外通过 VideoCaptureDataCallbak

接口把视频数据推送到上层模块。Media 模块的 WebRtcVideoCapturer 对象实现 VideoCaptureDataCallbak 接口，作为视频数据从 VideoCapture 模块向上推送的下一站。而 WebRtcVideoCapturer 则作为 VideoCaptureTrackSource 的成员变量，继续向上完成控制过程。

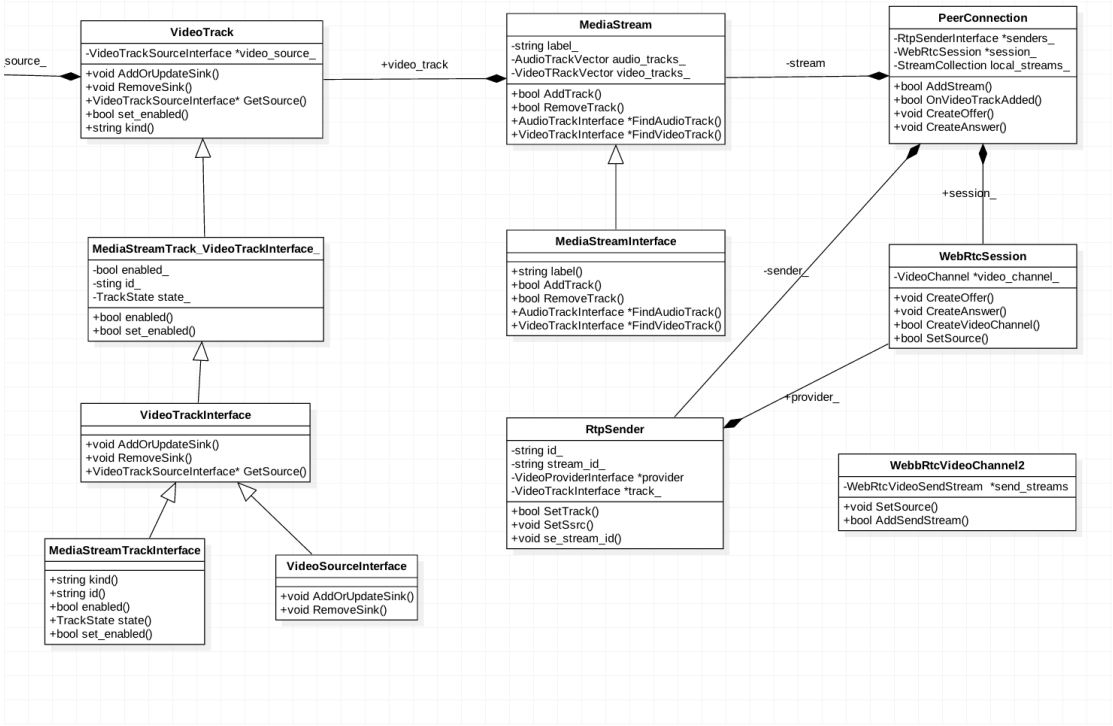

图 3 从 VideoTrack 到 PeerConnection

VideoCaptureTrackSource 作为数据源组合到 VideoTrack 中，而 VideoTrack 进一步组合到 MediaStream 中。MediaStream 最终会添加到 PeerConnection 中，通过 RtpSender 和 WebRtcSession 建立联系。WebRtcSession 在 SDP 创建 offer 和本地设置过程中通过 VideoChannel 把 WebRtcVideoCapturer 的数据流下一站设置到 WebRtcVideoSendStream 处，使得视频数据能够到达编码模块。

2.2 创建 MediaStream 并添加到 PeerConnection

我们知道，如果在 JS 端创建 WebRTC P2P 通信，需要首先创建 PeerConnection，然后通过 getUserMedia()得到本端的 MediaStream，接下来把 MediaStream 添加到 PeerConnection，最终执行 CreateOffer/OnSuccess 等 SDP 协商操作。在 Native 层，我们可以用以下代码实现这个过程。

```
Conductor::ConnectToPeer() {
    PeerConnectionFactoryInterface   *pcFactory = webrtc::CreatePeerConnectionFactory();
    PeerConnectionInterface *pc = pcFactory->CreatePeerConnection(config);
    VideoTrackInterface *videotrack = pcFactory->CreateVideoTrack("video_label",
                                pcFactory->CreateVideoSource(OpenVideoCaptureDevice()));
    MediaStreamInterface *stream = pcFactory->CreateLocalMediaStream("stream_label");
    stream->AddTrack(videotrack);
    pc->AddStream(stream);
    pc->CreateOffer();
}
Conductor::OnSuccesss(SessionDescriptonInterface *desc) {
    Pc->SetLocalDescription(desc);
    sendMessage(desc);
}
```

创建 MediaStream 是为了获取本端的音视频数据源，然后 MediaStream 设置到 PeerConnection 中，参与接下来的 SDP 协商过程。综合 2.1 节内容和上述代码，MediaStream 的创建和设置可由图 4 来表示。

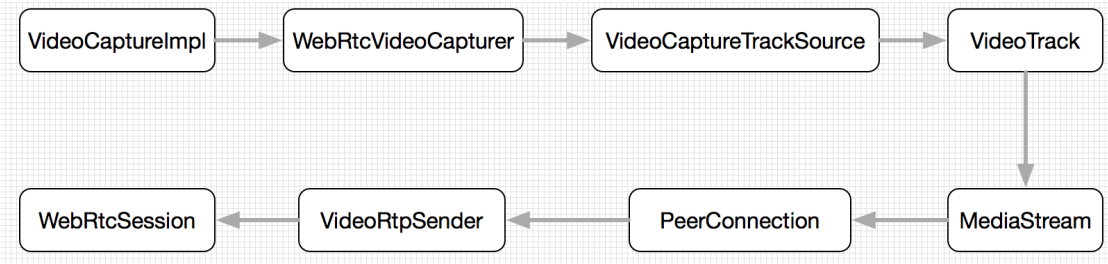

图 4 创建 MediaStream 并添加到 PeerConnection

当创建VideoTrack时，会以VideoCaptureTrackSource作为自己的videosource，而后者则进一步把 WebRtcVideoCapturer 作为自己的 videocapture。在WebRtcVideoCapturer对象中，VideoCaptureImpl作为module_成员。不同平台下的 VideoCaptureImpl 有不同的实现子类，如 Mac 平台下的实现类为VideoCaptureAvf。需要注意的是，当VideoCaptureTrackSource在初始化时，会调用WebRtcVideoCapturer::StartCapturing()函数开始视频采集工作。

VideoTrack 创建后，会被添加到 MediaStream，然后 MediaStream 被添加到PeerConnection，存储在 local_streams_ 成员变量中。接下来调用OnVideoTrackAdded()创建VideoRtpSender，该对象把VideoTrack、StreamLabel和 WebRtcSession 组合在一起。这样以来，底层的 VideoCaptureImpl 和WebRtcSession联系在一起，为接下来的SDP协商过程做好准备。

2.3 PeerConnection 的 CreateOffer 过程

CreateOffer是WebRTC建立P2P连接最重要的一步，它收集本地的音视频能力和网络层传输能力形成SDP描述结构。关于CreateOffer的详细过程分析是一项浩大工程，本文仅分析视频相关的CreateOffer过程，如图5所示。

PeerConnection在CreateOffer时，首先调用GetOptionsForOffer()函数获取本端的 MediaSession 描述信息，该信息描述了本端是否接收音视频数据、传输层选项、Stream 信息等。其中 Stream 信息是对 MediaStream 的描述，在

AddSendStreams()函数中，会获取 PeerConnection 中 RtpSender 对象的 MediaStream 信息，形成 Stream 结构后存储在 MediaSessionOptions 中。

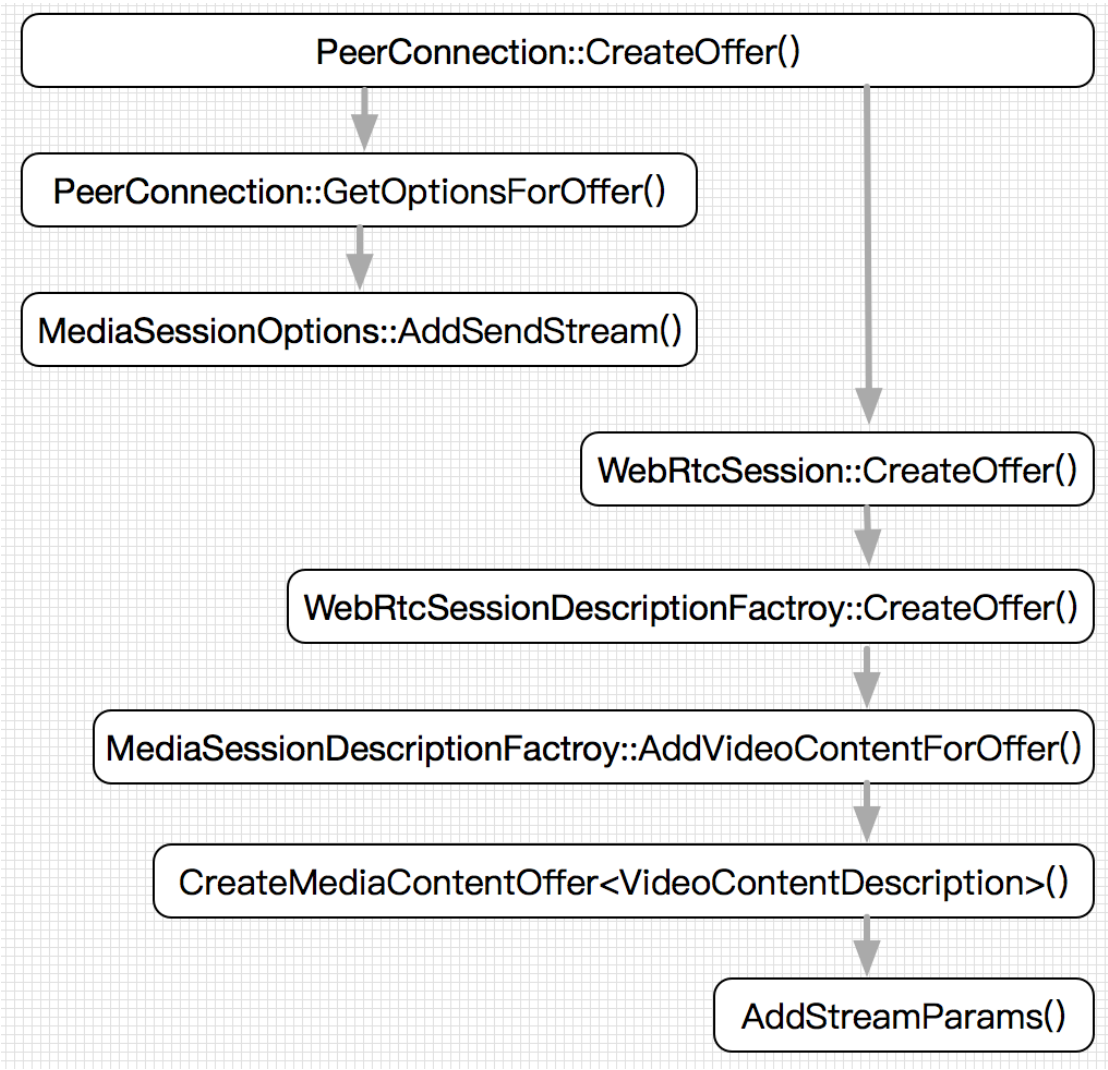

图 5 视频相关的 CreateOffer 过程

接下来 PeerConnection 把 CreateOffer 委托给 WebRtcSession，后者进一步委托给 WebRtcSessionDescriptionFactory，而最终的 SDP 创建工作都在 MediaSessionDescriptionFactory 中完成，其 CreateOffer()函数完成 Audio、Video 和 Data 的 MediaContent 生成工作。在 VideoContent 生成过程中，会根据 MediaSessionOptions 中的 Stream 信息生成 StreamParams 结构(该结构中包含

生成的 ssrc 信息)，存储在最终返回的 VideoContent 中。

至此，CreateOffer 操作完成，接下来会调用 PeerConnectionObserver 的 OnSuccess()函数执行 SDP 生成后的本地设置操作。

2.4 SetLocalDescription 过程

PeerConnection 在创建 CreateOffer 结束之后，会调用其 Observer 的 OnSuccess()接口进行回调。在该回调函数中，首先把生成的 offer 进行本地设置，然后把 offer 发送到 P2P 的对端。本节我们分析 offer 的本地设置过程，在这里将会最终建立发送端视频数据流水线，如图 6 所示。

PeerConnection 在进行 offer 本地设置时，首先在 WebRtcSession 中进行设置。在这里会根据 SDP 的内容创建 VideoChannel。VideoChannel 是 WebRtcSession 连接 VideoMediaChannel 的桥梁，而后者则是通向 VideoEngine 模块的通道。VideoChannel 创建完成以后，存储在 WebRtcSession 中。接下来 WebRtcSession 通过 PushDownLocalDescription()把 offer 内容向下设置到 VideoChannel，后者的 SetLocalContent_w()函数真正完成 offer 的本地设置工作。其中根据 offer 中关于 Stream 的描述信息调用 WebRtcVideoChannel2::AddSendStream()函数创建 WebRtcVideoSendStream 对象，这个对象是视频数据从 VideoCapture 模块出来后的下一站。注意该对象的存储以 ssrc 为索引，这个 ssrc 是在 CreateOffer 过程中为 Stream 对象生成的，代表了 VideoTrack。

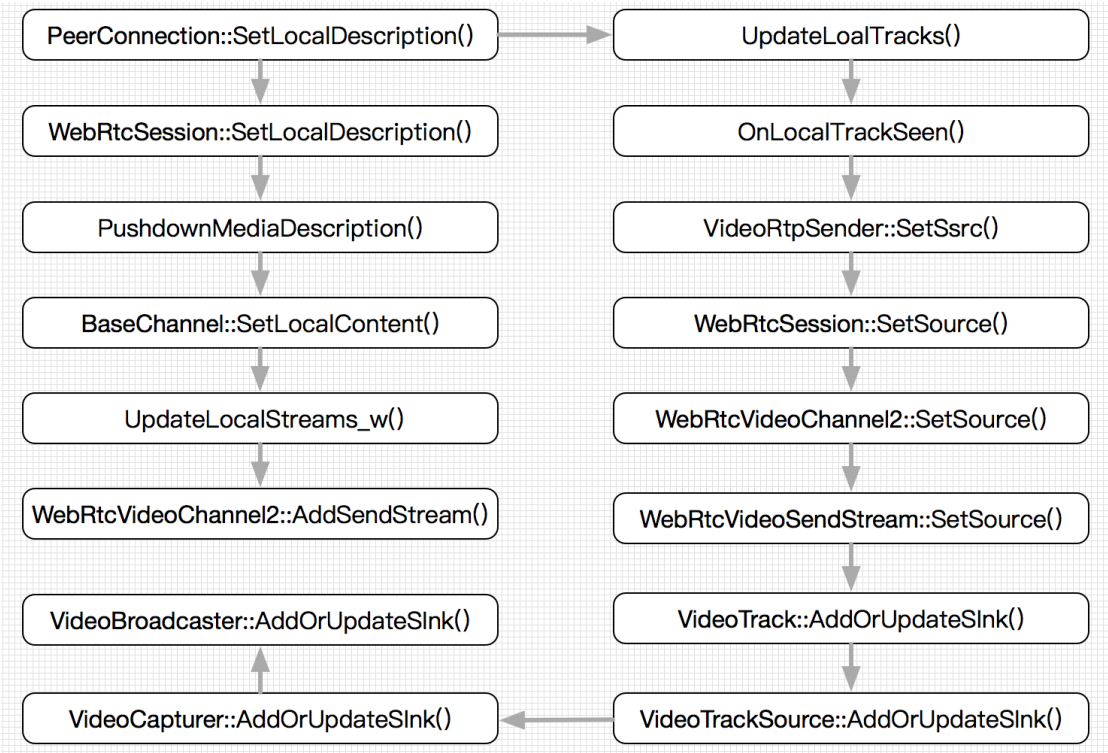

图 6 视频相关的 SetLocalDescription 过程

WebRtcSession 的本地设置工作完成以后，PeerConnection 以 offer 中的 Stream 为参数调用 UpdateLocalTracks()更新本地 MediaTrack，根据每个 Stream 中的 stream_label、track_id、ssrc、media_type 信息调用 OnLocalTrackSeen()函数。该函数最重要的功能是通过 track_id 找到 VideoRtpSender，然后设置其 ssrc。VideoRtpSender 调用和它关联的 WebRtcSession 进行下一步 SetSource(ssrc, track)设置操作。设置流程经过 VideoChannel 最终到达 WebRtcVideoSendStream，即后者的数据源是 VideoTrack。VideoTrack 通过 AddOrUpdateSink()反向设置自己的数据 Sink 是 WebRtcVideoSendStream，该设置过程会一路回溯直到 VideoBroadCaster。在那里完成设置后，VideoCapture 模块采集到的视频数据便会经过 OnFrame()接口到达 WebRtcVideoSendStream，进而到达 VideoEngine 模块，进行下一步的编码操作。

至此，PeerConnection 根据本地 MediaStream 建立视频数据流水线的过程分析完毕。这是一个复杂的过程，其重难点在于 CreateOffer 过程收集本地音视频信息描述以及接下来的本地 offer 设置过程。我们可以看到，ssrc 在 SDP 建立过程中发挥非常重要的作用。在随后的学习过程中，会针对 WebRTC 的 SDP 协商过程进行更深入全面的分析。

3 视频采集端数据流

通过上一节分析视频流水线的建立过程，接下来分析 VideoCapture 模块采集到的视频数据流的走向就很容易了。我们以 Mac 平台的 VideoCaptureAvf 为例进行分析，如图 7 所示。

VideoCaptureTrackSource 在初始化时已经开启视频采集工作，创建系统线程进行采集。在 Mac 平台上，视频数据从硬件采集后通过 RTCVideoCaptureAvfObjc 的 captureoutput 接口导出，经过初步处理后到达 VideoCaptureImpl 对象，该对象通过回调继续向上传递视频数据到达 WebRtcVideoCapturer。从此处开始线程从 Mac 系统线程切换到 Worker 线程。Worker 线程在对视频帧进行预处理之后，推送到 VideoBroadcaster 的 OnFrame()接口。在此处视频数据分两路处理：如果我们在之前添加了本地回显 VideoRender，则视频数据会进行本地显示；另一路数据则到达 WebRtcVideoSendStream，最终到达 VideoCaptureInput 对象处。视频数据会保存在 VideoCaptureInput 处，等待下一步 Encoder 线程来此拉取进行编码操作。

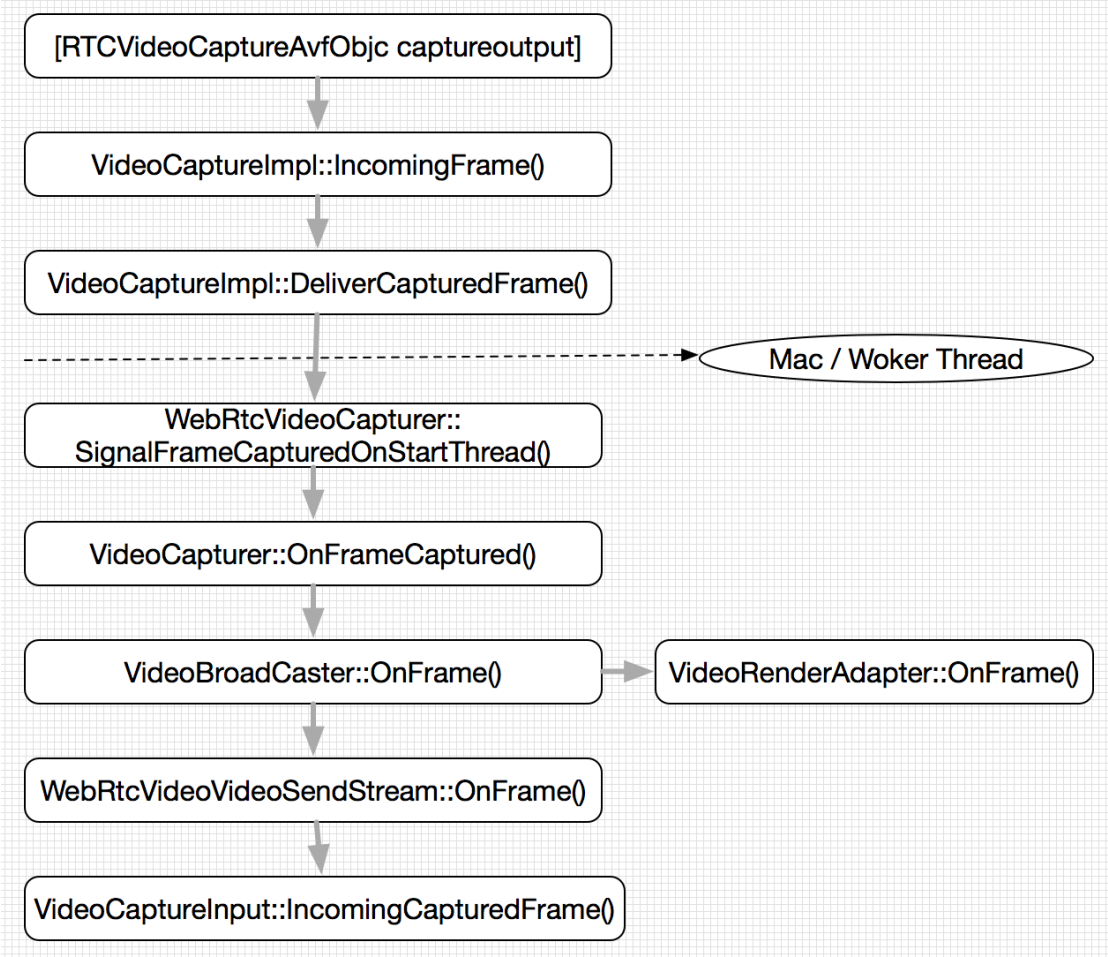

图 7 视频采集端数据流走向

至此，我们分析了视频数据从采集到待编码的流程。

4 总结

本文深入 WebRTC 视频采集模块的源代码，分析其平台无关部分和平台相关部分的实现细节，视频数据流水线的建立过程，和视频数据从采集到待编码的流向。其中重点分析了视频数据流水线的建立过程，对 WebRTC 的 SDP 协商过程有初步深入了解，为继续进一步学习 WebRTC 的技术原理奠定基础。

参考文献

[1] WebRTC 的模块处理机制：http://www.jianshu.com/p/9f4d4a725efb