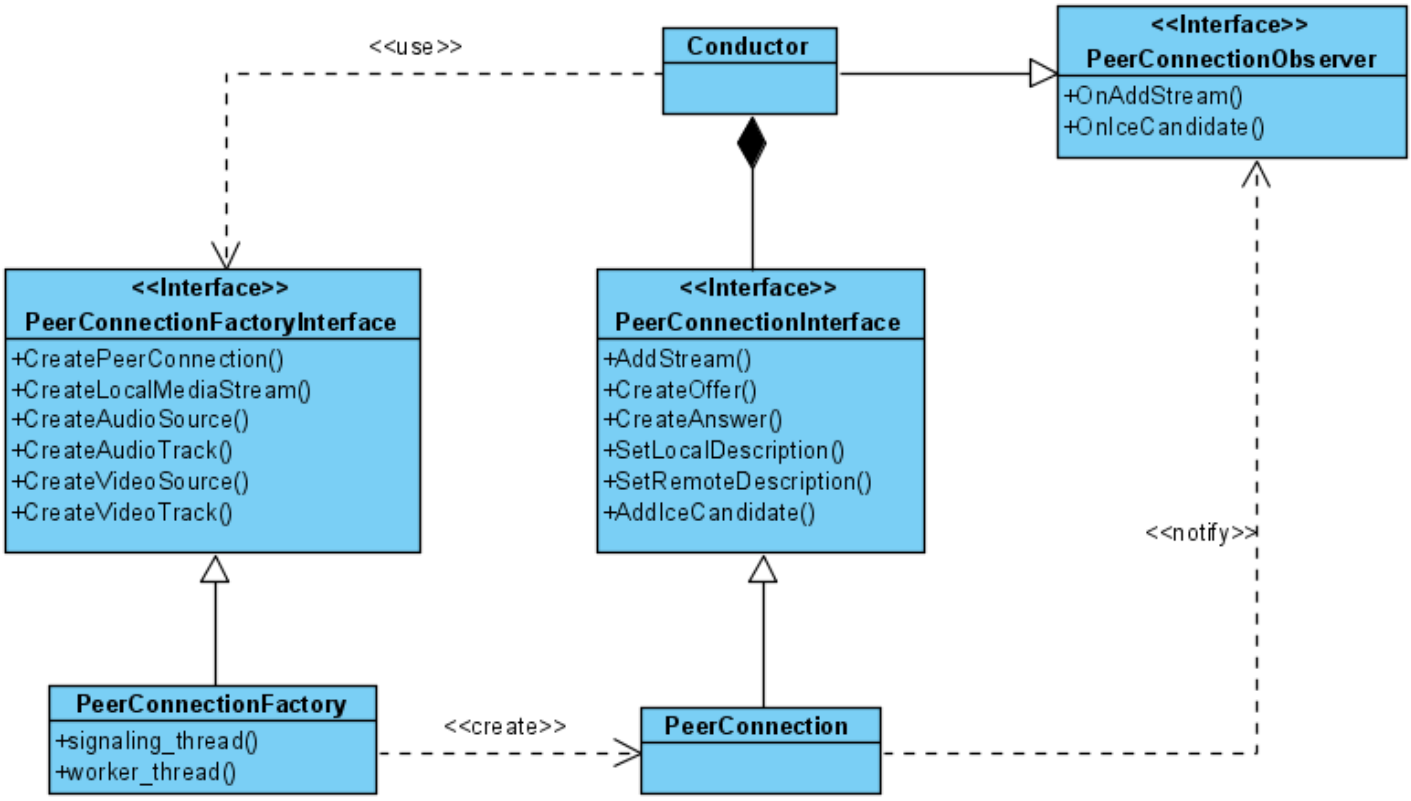
WebRTC手记之框架与接口 - 孤竹君 - 博客园

上一篇文章简单地介绍了下WebRTC的协议流程，这一篇就开始介绍框架与接口。

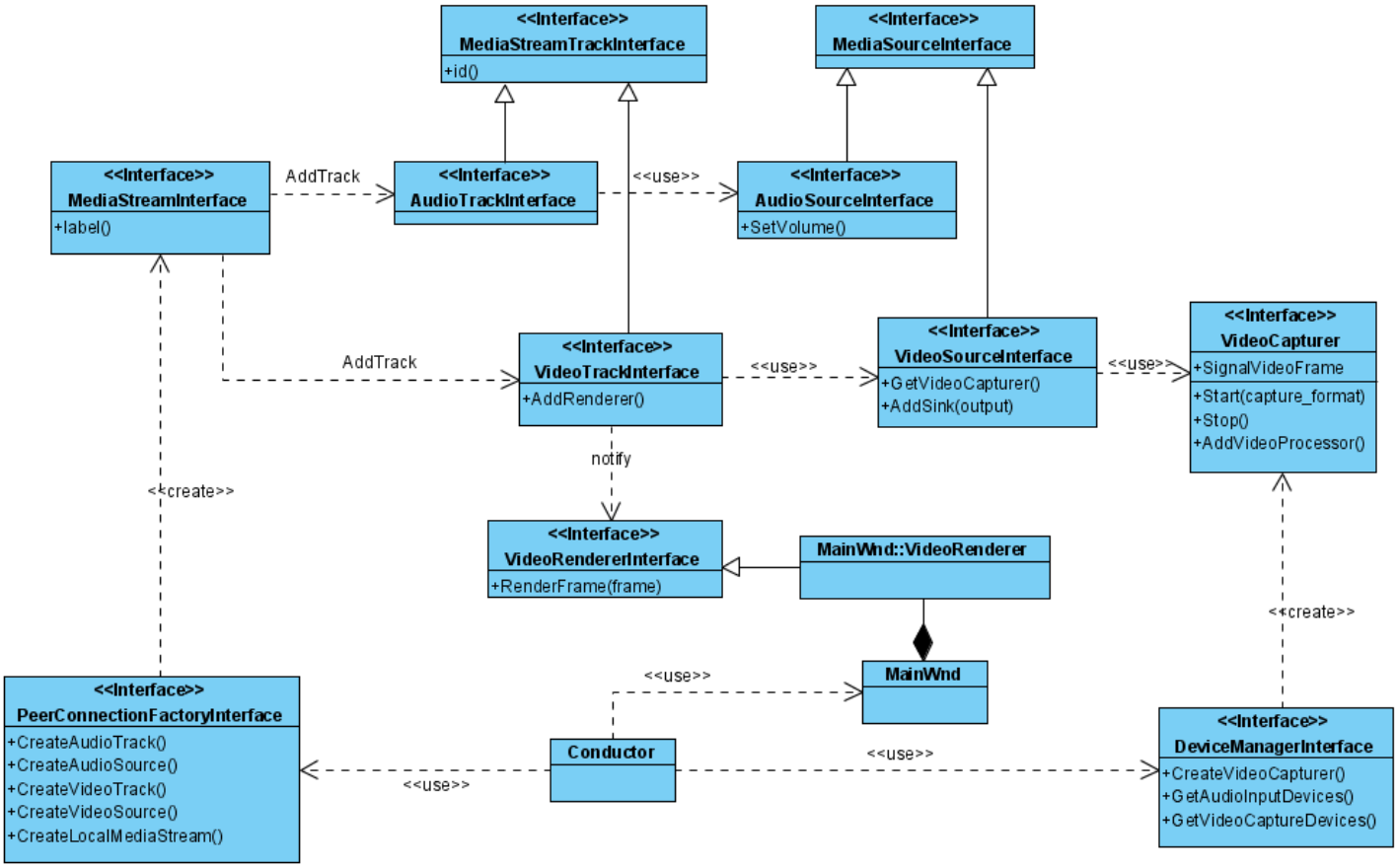
一提到框架，本能地不知道从什么地方入手了。曾经直接从Chromium项目对WebRTC的源码的集成方面入手，后来发现这个步子迈的太大了，看的越多，概念越混乱，看了半个月感觉也没啥沉淀。还是从WebRTC提供的示例工程peerconnection_client入手比较轻便。先抛开音视频流的构建和渲染流程，示例工程核心的代码结构如下：



从面向对象的视角来看，WebRTC的设计还是非常棒的，真正地做到了接口编程的概念，对WebRTC功能的使用都通过接口来进行，这样最大程度上保证了WebRTC模块的可定制性，这样就可以让WebRTC更多地回归到描述协议的本质。如果WebRTC对这些接口的实现不能满足你的业务需求，理论上你可以提供自己的实现逻辑。本图中的PeerConnectionFactoryInterface和PeerConnectionInterface没有这种定制的代表性，因为重新提供它们的实现逻辑的需求场景基本上不存在（即便不用重写，但也支持参数的定制，具体请参见CreatePeerConnectionFactory的重载方法）。但是音视频相关的接口定制的场景就很普遍了，比如Chromium浏览器集成WebRTC，但是音视频采集需要走Chromium自己的音视频模块，所以Chromium对WebRTC音视频的采集接口重新做了实现适配，以后有机会肯定非常乐意分享下Chromium源码对WebRTC的集成，不过那也是在对WebRTC熟悉完之后的工作了。

图中Conductor是该示例工程提供的核心业务类，整个WebRTC的使用都浓缩在这个类中。Conductor通过CreatePeerConnectionFactory方法创建了一个PeerConnectionFactoryInterface接口的实现对象，通过这个接口，可以创建关键的PeerConnectionInterface接口，PeerConnectionInterface接口是WebRTC的协议核心。此外，PeerConnectionFactoryInterface接口还提供了创建本地音视频流的功能接口，这个部分稍后再述。根据图中PeerConnectionInterface接口的成员方法可以看出，WebRTC通信流程的交互接口基本上都在这里面了，给Conductor的回调通知是通过PeerConnectionObserver接口来完成。具体的交互流程请参见上一篇博文。

接下来分析本地音视频的相关接口，由于音视频内容较多，这里先介绍下接口概念，不谈具体实现（下一节专门讲解WebRTC原生的音视频采集），还是以peerconnection_client工程为例：



这里涉及到非常多的音视频相关接口，基本上都是概念性的，最怕遇到新的设计概念，主要是怕自己理解有误差，下面谈一下我对这些接口概念的理解：

MediaStream概念： 表示媒体流，由MediaStreamInterface接口抽象，每个媒体流都有一个唯一的标识（通过label成员方法返回），它由一系列的音频Track（由AudioTrackInterface接口抽象）和视频Track组成（由VideoTrackInterface接口抽象）。

Track概念：具体指上图结构中AudioTrackInterface和VideoTrackInterface接口，Track表示的是在媒体流中轨的概念，AudioTrackInterface标识的是音频轨，VideoTrackInterface标识的是视频轨，一个MediaStreamInterface标识的媒体流中允许携带多个媒体轨数据，它们之间是独立的，在编码渲染的流程中各自处理。如果概念还很模糊，轨的概念就相当于音频数据中的声道概念（左声道、右声道）、视频数据中的YUV场的概念。Track既然封装了媒体轨数据，那就必然有个媒体源做为数据的提供者，如音频Track由AudioSourceInterface接口作为数据源提供者，视频Track由VideoSourceInterface接口作为数据的提供者。有输入接口就必然有输出接口，这部分在该图中只展示了视频数据的输出接口VideoRendererInterface，这里的Render命名的意思并不是仅限于将视频和音频数据渲染出来，应该理解成输出接口。

VideoSourceInterface：抽象视频源接口供VideoTracks使用，同一个源可以被多个VideoTracks共用。视频源接纳了一个VideoCapturer接口，抽象视频采集的逻辑，我们可以提供自己的VideoCapturer实现做为视频数据的采集源。VideoCapturer是个关键的定制接口，比如Chromium源码就是自己实现了VideoCapturer接口而没用原生的WebRTC采集实现，但Chromium的音视频采集都在browser进程，因此它对VideoCapturer接口的实现要比想象的复杂，它需要从主进程接收到视频帧数据然后触发VideoCapturer的SignalFrameCaptured信号。

AudioSourceInterface：概念上同VideoSourceInterface类似，抽象音频源接口供AudioTracks使用，但是从源码中理解，这是个伪概念，因为没有提供一个类似于VideoCapturer的AudioCapturer接口，这里没有音频的采集逻辑，实际上WebRTC的音频采集接口使用的是AudioDeviceModule，在创建PeerConnectionFactory的时候可以由外界定制，如果没有，则内部创建AudioDeviceModuleImpl来实现此接口完成音频设备的采集工作。可能是功力不够，反正我是不太理解音频采集和视频采集这种设计的不对称性。如果也封装一个AudioCapturer接口的概念，这样可定制性是不是可以更高。

构建媒体流的过程基本上就是构建Video Track和Audio Track，并将其添加到Media Stream里。在peerconnection_client工程中，Conductor依赖DeviceManagerInterface接口的CreateVideoCapturer方法创建一个

当前可用的视频设备采集对象VideoCapturer，将它作为视频采集源中的数据来源（通过挂接VideoCapturer的SignalVideoFrame信号来接收视频数据），此外MainWnd还创建了一个内部类VideoRenderer从VideoRendererInterface接口派生，并将其添加到Video Track中， VideoRenderer的实现就是将接收到的视频帧数据渲染到窗口上。

下一篇开始分析WebRTC原生的音视频本地采集模块。