Chapter 5

MediaStreamAPI (còn được gọi là **getUserMediaAPI**)

Các MediaStreamAPI đại diện cho đồng bộ dòng phương tiện truyền thông. Ví dụ: một luồng được lấy từ đầu vào máy ảnh và micrô đã đồng bộ hóa các bản nhạc và video.

Có lẽ cách dễ nhất để hiểu MediaStreamAPI là xem xét nó một cách tự nhiên:

1.Trong trình duyệt của bạn, điều hướng đến các mẫu **WebRTCgetUserMedia** .

2.Mở bảng điều khiển.

3.Kiểm tra stream biến nằm trong phạm vi toàn cầu.

Mỗi cái đều MediaStream có một đầu vào, có thể là một MediaStreamđược tạo bởi **getUserMedia()** và một đầu ra, có thể được chuyển cho một phần tử video hoặc một **RTCPeerConnection**.

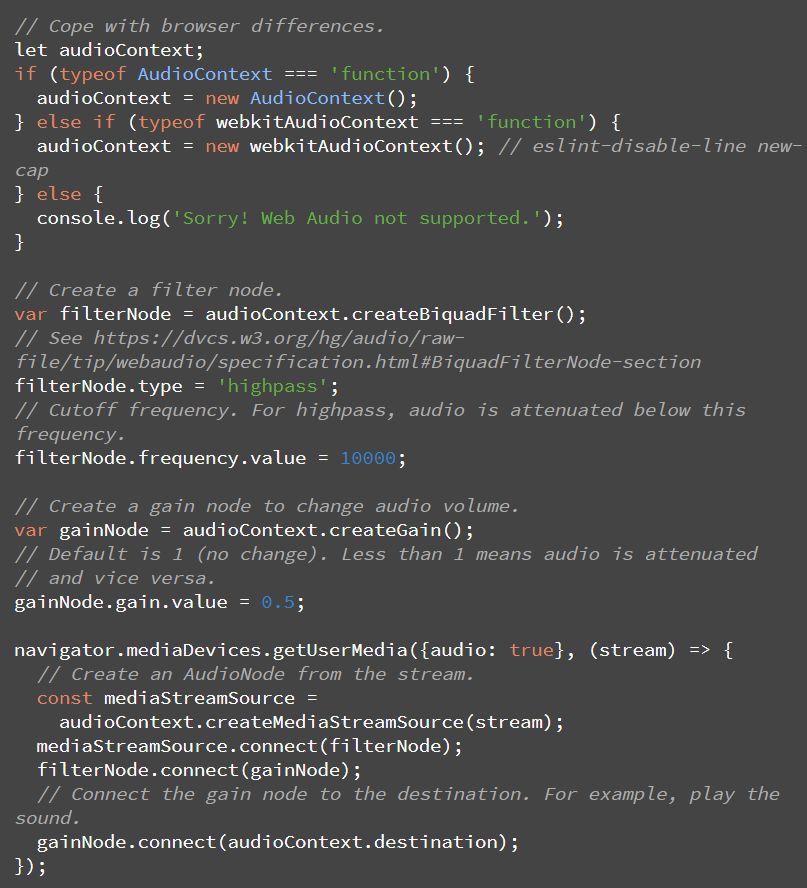
Các **getUserMedia()** phương pháp có một MediaStreamConstraints đối tượng tham số và trả về một Promise mà giải quyết cho một MediaStream đối tượng.

Mỗi MediaStream có một label, chẳng hạn như 'Xk7EuLhsuHKbnjLWkW4yYGNJJ8ONsgwHBvLQ'. Một mảng MediaStreamTracks được trả về bởi các phương thức getAudioTracks() và getVideoTracks().

Ví getUserMediadụ, **stream.getAudioTracks()** trả về một mảng trống (vì không có âm thanh) và giả sử một webcam đang hoạt động được kết nối, **stream.getVideoTracks()** trả về một mảng MediaStreamTrack đại diện cho luồng từ webcam. Mỗi kênh MediaStreamTrack có một loại ( 'video' hoặc 'audio'), một label (ví dụ như 'FaceTime HD Camera (Built-in)') và đại diện cho một hoặc nhiều kênh âm thanh hoặc video. Trong trường hợp này, chỉ có một đoạn video và không có âm thanh, nhưng có thể dễ dàng hình dung các trường hợp sử dụng có nhiều hơn, chẳng hạn như ứng dụng trò chuyện lấy luồng từ camera trước, camera sau, micrô và một ứng dụng chia sẻ màn.

Một MediaStream có thể được gắn vào phần tử video bằng cách đặt srcObjectthuộc tính. Trước đây, điều này đã được thực hiện bằng cách đặt src thuộc tính thành URL đối tượng được tạo bằng **URL.createObjectURL(),** nhưng hiện nay phương pháp này không còn được sử dụng nữa

getUserMediacũng có thể được sử dụng làm nút đầu vào cho API âm thanh web :



Các ứng dụng và tiện ích mở rộng dựa trên Chromium cũng có thể kết hợp getUserMedia. Thêm **audioCapture** và / hoặc **videoCapture** quyền vào tệp kê khai cho phép quyền được yêu cầu và cấp chỉ một lần khi cài đặt. Sau đó, người dùng không được yêu cầu quyền truy cập máy ảnh hoặc micrô.

Quyền chỉ phải được cấp một lần cho **getUserMedia().** Lần đầu tiên, nút Cho phép được hiển thị trên thanh thông tin của trình duyệt . Quyền truy cập HTTP cho **getUserMedia()** đã bị Chrome ngừng sử dụng vào cuối năm 2015 do tính năng này được phân loại là Tính năng mạnh.

Mục đích có khả năng cho phép MediaStream bất kỳ nguồn dữ liệu phát trực tuyến nào, không chỉ máy ảnh hoặc micrô. Điều này sẽ cho phép phát trực tuyến từ dữ liệu được lưu trữ hoặc các nguồn dữ liệu tùy ý, chẳng hạn như cảm biến hoặc các đầu vào khác.

**Constraints (Ràng buộc)**

Ràng buộc có thể được sử dụng để đặt giá trị cho độ phân giải video **getUserMedia().** Điều này cũng cho phép hỗ trợ các ràng buộc khác , chẳng hạn như tỷ lệ khung hình; chế độ camera (camera trước hoặc sau); tỷ lệ khung hình, chiều cao và chiều rộng; và một **applyConstraints()**.

Việc đặt giá trị hạn chế không được phép đưa ra một DOMExceptionhoặc một OverconstrainedErrornếu, ví dụ, một nghị quyết yêu cầu không có sẵn. Để xem điều này thực tế, hãy xem các mẫu WebRTC **getUserMedia**: chọn độ phân giải cho bản trình diễn.

**Screen and tab capture (màn hình và tab)**

Ứng dụng Chrome cũng giúp bạn có thể chia sẻ video trực tiếp của một tab trình duyệt hoặc toàn bộ màn hình thông qua **chrome.tabCapture** và **chrome.desktopCapture** các API.

Cũng có thể sử dụng chụp màn hình làm **MediaStream** nguồn trong Chrome bằng cách sử dụng **chromeMediaSource** ràng buộc thử nghiệm . Lưu ý rằng chụp màn hình yêu cầu HTTPS và chỉ nên được sử dụng để phát triển do nó được bật thông qua cờ dòng lệnh.

**Signaling: Session control, network, and media information (Báo hiệu: Kiểm soát phiên, mạng và thông tin phương tiện)**

WebRTC sử dụng **RTCPeerConnection** để giao tiếp dữ liệu trực tuyến giữa các trình duyệt (còn được gọi là ngang hàng (peers)), nhưng cũng cần một cơ chế để điều phối giao tiếp và gửi các thông điệp điều khiển, một quá trình được gọi là báo hiệu. Các phương pháp và giao thức báo hiệu không được WebRTC chỉ định. Báo hiệu không phải là một phần của **RTCPeerConnection** API.

Thay vào đó, các nhà phát triển ứng dụng WebRTC có thể chọn bất kỳ giao thức nhắn tin nào họ thích, chẳng hạn như SIP hoặc XMPP và bất kỳ kênh giao tiếp song công (hai chiều) nào thích hợp. Các appr.tc dụ sử dụng XHR và API Kênh là cơ chế truyền tín hiệu. Bộ mã sử dụng Socket.io chạy trên máy chủ Node .

-Báo hiệu được sử dụng để trao đổi ba loại thông tin:

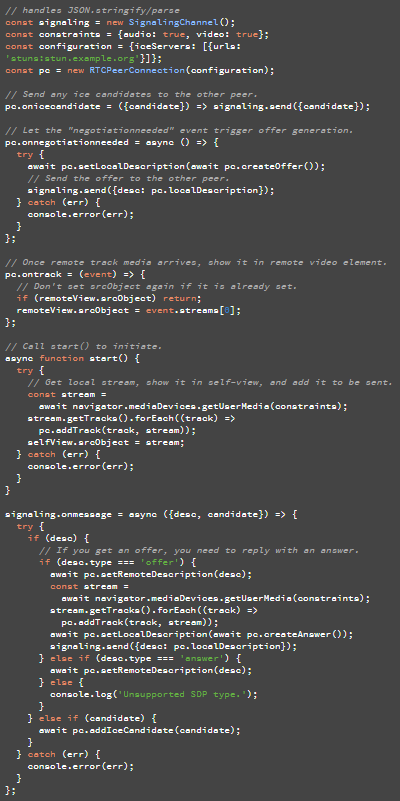
1.Thông báo điều khiển phiên: để khởi tạo hoặc đóng giao tiếp và báo lỗi.

2.Khả năng phương tiện: codec và độ phân giải nào có thể được xử lý bởi trình duyệt của bạn và trình duyệt mà nó muốn giao tiếp?

3.Việc trao đổi thông tin thông qua báo hiệu phải hoàn tất thành công trước khi phát trực tuyến ngang hàng có thể bắt đầu.

=>**Việc trao đổi thông tin thông qua báo hiệu phải hoàn tất thành công trước khi phát trực tuyến ngang hàng có thể bắt đầu.**

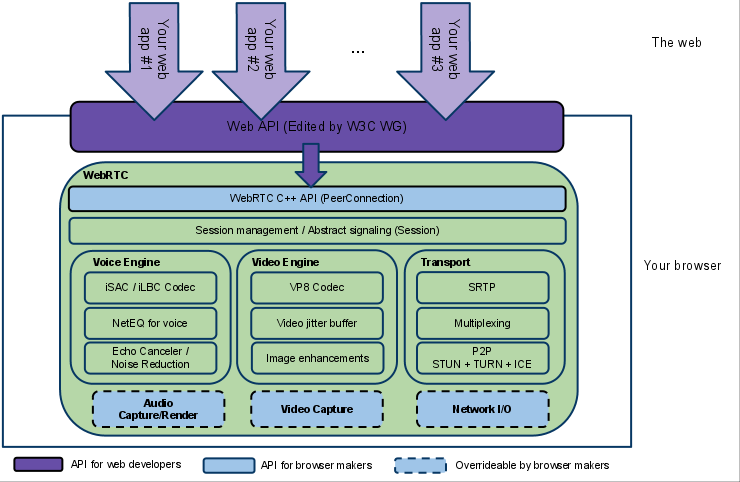
Ví dụ: Mã giả định sự tồn tại của một số cơ chế báo hiệu được tạo trong createSignalingChannel()phương thức. Cũng lưu ý rằng trên Chrome và Opera, RTCPeerConnectionhiện đã có tiền tố.



**RTCPeerConnection**

**RTCPeerConnection** là thành phần WebRTC xử lý giao tiếp ổn định và hiệu quả về truyền dữ liệu.

Sơ đồ kiến ​​trúc WebRTC cho thấy vai trò của **RTCPeerConnection**:



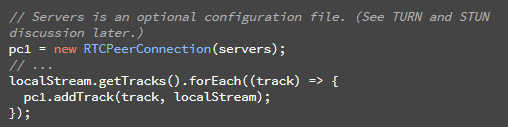
RTCPeerConnection without server (RTCPeerConnection không có máy chủ)

Đoạn mã sau được lấy từ các mẫu WebRTC Kết nối ngang nhau, có kết nối cục bộ và từ xa **RTCPeerConnection** (và video cục bộ và từ xa) trên một trang web. người gọi và người được gọi nằm trên cùng một trang, nó làm cho hoạt động của **RTCPeerConnectionAPI** rõ ràng hơn một chút vì các RTCPeerConnection đối tượng trên trang có thể trao đổi dữ liệu và thông điệp trực tiếp mà không cần phải sử dụng cơ chế báo hiệu trung gian.

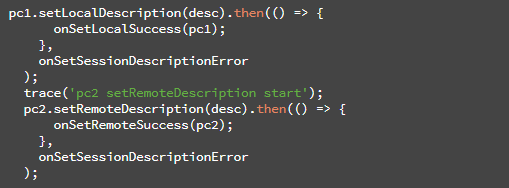
Trong ví dụ này, pc1đại diện cho ngang hàng cục bộ (người gọi) và pc2đại diện cho ngang hàng từ xa (callee).

**1.Người gọi:**

Tạo **RTCPeerConnection** và thêm luồng từ **getUserMedia()**:

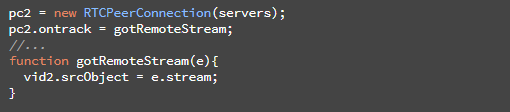


Tạo và đặt nó làm mô tả cục bộ cho pc1và làm mô tả từ xa cho pc2. Điều này có thể được thực hiện trực tiếp trong mã mà không cần sử dụng tín hiệu vì cả người gọi và người gọi đều ở trên cùng một trang:



**2.Người được gọi:**

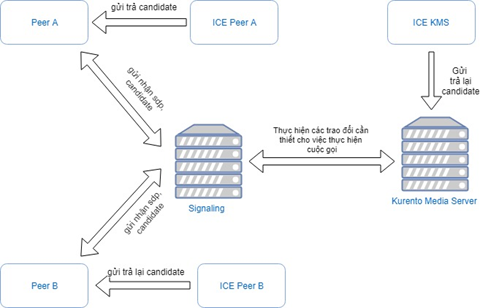
Tạo pc2 và khi luồng từ pc1 được thêm vào, hiển thị nó trong phần tử video:



II.Ứng dụng trong video call bằng web RTC:

**I. Các thành phần tham gia vào thực hiện ứng dụng video call**

Với các ứng dụng WebRTC thông thường, việc thực hiện các ứng dụng media sử dụng API WebRTC cung cấp thông thường hầu như khá phức tạp, rất khó để cho lập trình viên. Như đã đề cập đến ở chương 2, kurento cung cấp một số API nhằm mục đích đơn giản hóa việc xây dựng các ứng dụng đa phương tiện, một phần trong các API này dựa vào WebRTC API có sẵn. Một điều nữa là mặc dù các ứng dụng WebRTC chưa hẳn là peer-to-peer, nó vẫn cần một server đứng giữa để thực hiện việc trao đổi các thông tin cần thiết giữa các peer, có thể cần một media server cho việc nâng cao hiệu suất của ứng dụng WebRTC. Dưới đây là một số thành phần tham gia vào khi thực hiện cuộc gọi.



Tóm lượt lại hình trên ta có thể thấy có 4 thành phần chính tham gia vào quá trình thực hiện. Dưới đây là mô tả tóm gọn các thành phần:

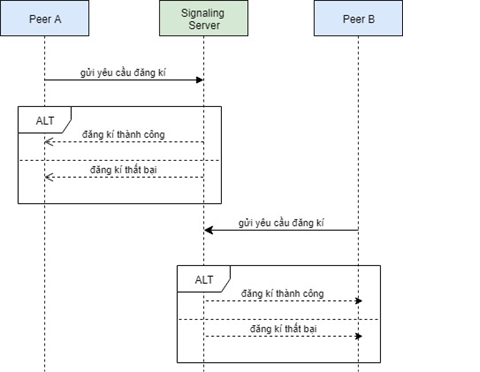
* Peer: Chính là đối tượng kết nối ( ở đây có thể là trình duyệt web,mobile…).
* Signaling Server: Như đã giới thiệu ở các phần trên, ở đây nó đứng giữa thực hiện trao đổi với các peer và KMS.
* ICE (Interactive Connectivity Establishment): Quá trình hỗ trợ vượt NAT, nó sẽ nhận đăng kí từ các peer và trả về candidate (là các cách để kết nối với peer đó). ICE ở đây có thể là tập hợp bao gồm STUN, TURN server được đặt ở các đối tượng là peer và KMS. Phần này đã được giới thiệu ở phần trước

Các đối tượng này sẽ tham gia vào trong quá trình thực hiện cuộc gọi video. Về cơ bản ta có thể chia quá trình này thành 3 quá trình con là đăng kí, thực hiện cuộc gọi,kết thúc cuộc goi.Chi tiết về quá trình thực hiện sẽ được mô tả bên dưới.

**II. Quá trình thực hiện ứng dụng video call**

**1. Đăng kí người dùng**

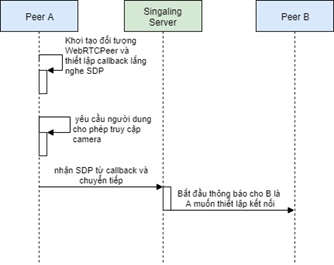
Bước này thực hiện khá đơn giản với việc có 3 tác nhân là người dùng A, người dùng B và signaling server. Server này sẽ là trung gian xử lý các yêu cầu đăng kí của hai bên (ví dụ có thể lưu vào database). Nó cũng đưa ra các quyết định liệu có thể cho người dùng đăng kí hay không. Có hai trạng thái mà nó gửi về cho người dùng ở bước này là đăng kí thành công và đăng kí thất bại. Hình dưới mô tả pha đăng kí người dùng.



**2.Thực hiện cuộc gọi**

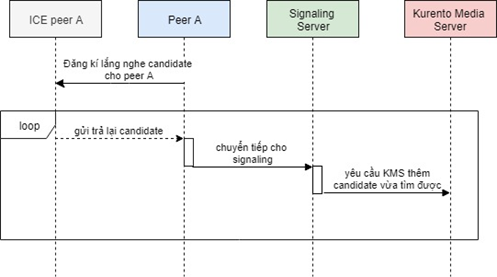
Bước chủ yếu được giới thiệu ở đây là thực hiện cuộc gọi. Ở bước này các peer sẽ phải thực hiện một số trao đổi nhất định để có thể đạt được kết nối với KMS. Các trao đổi này được trung gian qua Signaling Server và lấy được candidate thông qua các ICE đứng bên cạnh các đối tượng. Dưới đây là mô tả chi tiết các quy trình để thực hiện một cuộc gọi video.

**Quy trình lấy được SDP**

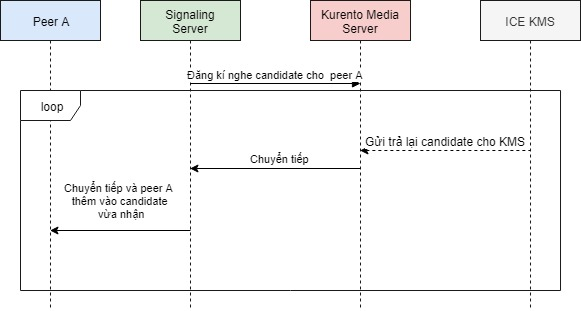


Khi thực hiện việc đăng kí thành công, để bắt đầu thiết lập cuộc gọi, mỗi bên cần biết một thông số nhất định. Đầu tiên, bắt đầu khởi tạo một đối tượng là WebRtcPeer ( đã nhắc qua ở chương 2).Sau đó ta bắt đầu thiết lập một callback để lắng nghe SDP offer được gửi về. Khi nhận được SDP thành công, Peer A bắt đầu gửi yêu cầu cấp quyền truy cập một số tài nguyên thiết bị tới người dùng.Nếu người dùng đồng ý, chúng ta bắt đầu thực hiện quá trình tiếp theo là gửi các thông tin ban đầu từ Peer A ( như là SDP, đến từ đâu, muốn kết nối tới đâu…) đến signaling.Signaling có nhiệm vụ lưu lại các thông tin này và bắt đầu thông báo tới trình duyệt B là có ai đó muốn kết nối.Khi Peer B nhận được thông báo bên A muốn kết nối, nó sẽ thiết lập tương tự như Peer A khi lấy SDP offer và truy cập thiết bị. Khi đó Signaling sẽ có thông tin của hai Peer và bắt đầu vào quá trình thiết lập cuộc gọi với KMS.

**Trao đổi ứng viên (candidate)**



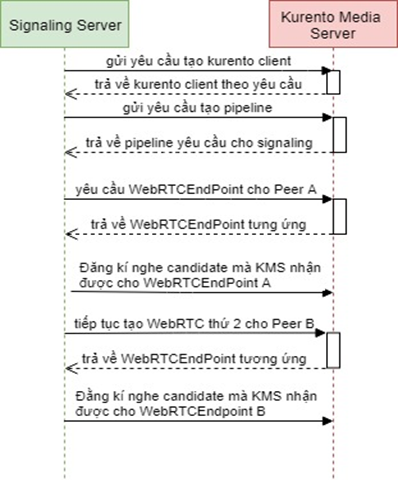
Quá trình này song song với quá trình đạt được SDP, mục đích của quá trình này nhằm mở ra các cách khác nhau mà cách peer khác có thể kết nối với A.Đầu tiên ta cần làm là đăng kí lắng nghe candidate tới ICE peer A bằng cách ta sẽ thiết lập một callback là OnIceCandidate ( đây là một option khi tạo đối tượng WebRtcPeer).Khi đó,ICE cho trình duyệt A sẽ thực hiện chức năng của nó là tìm các cách để có thể kết nối với A. Khi tìm được, nó sẽ gửi lại cho trình duyệt A thông qua hàm callback OnIceCandidate.Bước tiếp theo trình duyệt A gửi lại cho signaling server, server này đơn giản là ủy quyền lại cho Kurento Media Sever (KMS) bằng cách gọi addCandidate để yêu cầu thêm vào các cách để kết nối với A (candidate).Vì vậy KMS sẽ biết cách để kết nối với peer A.Việc này tương tự với Peer B.



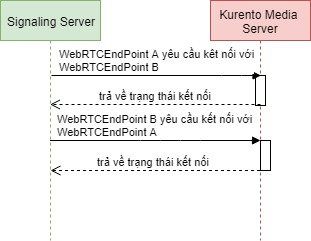
Quá trình này được thực hiện khi mà WebRtcEndpoint đã được tạo ra bên trong KMS. Khi đó ta cần biết cách để biết làm sao để kết nối với WebRtcEndpoint tương ứng. Bằng cách thêm vào một máy chủ ICE bên cạnh KMS (có thể là STUN hoặc TURN). Nó sẽ nói cho KMS biết làm cách nào để kết nối với WebRtcEndpoint trong đó. Từ đó KMS sẽ gửi các cách kết nối với WebRtcEndpoint này (candidate) trở lại chỗ mà nó đã đăng kí mỗi khi mà máy chủ ICE gửi trả về một candidate tương ứng. Các candidate này được gửi trở lại các peer qua Signaling làm trung gian. Các peer sẽ thực hiện công việc cuối cùng là thêm cái candidate đó vào (bằng cách thực hiện addCandidate ở đối tượng WebRtcPeer) và quá trình này có thể sẽ phải lặp lại nhiều lần. Nếu quá trình trao đổi thành công, Peer sẽ biết cách để kết nối với WebRtcEndpoint trong KMS vừa tạo. Kết quả từ hai quá trình kể trên là các Peer và KMS đã hiểu cách kết nối với nhau.

**Bắt đầu thiết lập cuộc gọi**

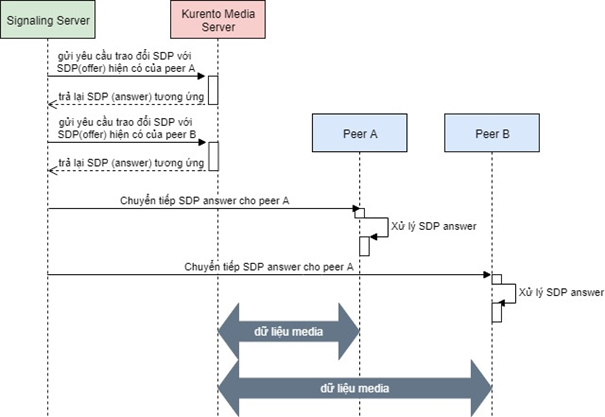
Ngay sau khi B chấp nhận kết nối, chúng ta bắt đầu thiết lập các bước cần thiết để có thể đạt được cuộc gọi giữa hai. Trọng phần này các bước sẽ được thực hiện ở Signaling, dưới đây là mô tả các bước Signaling cần thực hiện để thiết lập cuộc gọi.



Ở bước này đầu tiên ta cần phải tạo một kurentoClient bằng cách gọi hàm kurento ( được kurento cung cấp ) nhằm kết nối tới KMS.Sau khi đã kết nối được với KMS,chúng ta cần gửi yêu cầu đến tạo một pipeline, chúng ta sử dụng đối tượng kurentoClient đã tạo trước đó gọi hàm create (được thiết kế để tương tác với KMS như đề cập ở chương trước) với tham số là MediaPipeline.Tiếp đó, ngay sau khi có được pipeline, chúng ta cần hai WebRTCEndpoint cho hai trình duyệt tương ứng A và B. Mục đích của việc này là để các luồng media hoạt động trên KMS có thể trao đổi với nhau.



Mặc dù đã tạo hai WebRTCEndPoint nhưng chúng ta cần phải làm một điều là kết nói chúng lại với nhau, việc kết nối này được thực hiện trên cả hai WebRTCEndPoint. Thực hiện điều này bằng cách sử dụng đối tượng hai đối tượng WebRTCEndPoint ở trên và thực hiện gọi invoke với tham số là ‘connect’. Việc này được thực hiện lần lượt với cả hai WebRTCEndPoint.



Bước cuối cùng trong việc thực hiện thiết lập cuộc gọi là việc trao đổi SDP với KMS.Bên phía signaling đã có hai SDP offer được gửi cho nó từ lần trước.Cái signaling làm là gọi sử dụng đối tượng WebRTCEndPoint mà KMS đã trả về từ trước, gọi hàm processOffer với tham số là SDP offer tương ứng với mỗi peer và một callback.Khi thực hiện điều này, bên phía KMS sẽ đáp lại SDP answer và gửi trả về tương ứng với mỗi peer.Sau khi nhận được SDP answer, mỗi peer sẽ gọi processAnswer để xử lý SDP answer được trả về. Từ đây mỗi peer có thể tương tác với nhau, gửi/nhận dữ liệu media qua KMS.

**Kết thúc cuộc gọi**

Pha cuối cùng trong quy trình là ngừng cuộc gọi. Khi có yêu cầu ngừng từ một peer, peer này sẽ kết thúc phiên tương tác bằng cách ngắt kết nối camera và gửi yêu cầu tới signaling, signaling sẽ tiếp tục gửi lại yêu cầu này tới peer đang tương tác và thực hiện tương tự. Tiếp theo signaling sẽ gửi tới KMS yêu cầu ngắt kết nối với hai peer và giải phóng tài nguyên, kết thúc phiên làm việc. Hình dưới mô tả phiên làm việc kết thúc tương tác.

