Ver. 2020

웹에서의 실시간 비디오, 오디오, 데이터통신 프로그래밍을 위한

W3C WebRTC 표준해설서

TTA 한국정보통신기술협회
Telecommunications Technology Association

머리말

2020년 신종 코로나바이러스감염증-19(COVID-19) 사태로 언택트(비대면) 기술 확산과 함께 산업의 중심이 오프라인에서 온라인으로 옮겨지면서 ICT의 중요성이 더욱 커지고 있습니다. 정부는 포스트 코로나 시대에 대응하여 '한국판 뉴딜' 정책을 발표로 '디지털 뉴딜', '그린 뉴딜', '안정망 강화'의 대형 프로젝트를 진행하여, D.N.A 생태계 강화, 비대면 사업육성, SOC 디지털화 등 코로나 이후의 위기를 극복하기 위해 디지털 국가로의 발전 정책을 추진 중에 있습니다.

D.N.A(Data-Network-AI)를 기반으로 디지털 국가와 스마트한 정부, 국토, 산업을 목표로 더 안전하고 편리해진 국민들의 삶 조성을 위하여 데이터댐, AI, 스마트의료, 디지털트윈(Digital Twin) 등의 ICT 기반 기술 발전과제를 선정하였습니다. ICT 표준은 R&D와 시장을 연결하는 핵심수단으로, ICT 분야의 기술적 상용화와 사용자 수요에 맞는 기술 개발을 위한 ICT 표준화를 추진하고 있습니다.

한국정보통신기술협회는 ICT 국제표준화전문가 활동지원을 통해 국내 산업체의 ICT 시장 경쟁력을 제고하고 국제표준화기구에서의 한국 영향력 확대를 위해 노력하고 있습니다. ICT 표준화포럼에서는 국내 산업체 등 중소기업의 수요를 반영하여 IETF의 I2NSF(Interface to Network Security Functions), W3C의 RTC(Real-time Communication Between Browsers), DID(Decentralized ID), Vehicle cloud(Vehicle Information Service Specification)의 최신 표준 4개를 선정하고, 관련 표준의 해설을 담은 ICT 국제표준화전문가 표준해설서를 마련하였습니다.

본 표준해설서는 해당 기술의 구현 및 지식습득이 필요한 국내 산업체와 ICT 국제표준전문가 표준화 활동에 교과서적인 전문적 지식전달 목적으로 활용될 수 있을 뿐만 아니라, ICT 사실표준화의 국내 산업 활성화에 기여할 수 있을 것으로 기대합니다. 마지막으로 표준해설서 발간에 참여해주신 사실표준화기구 미러포럼 표준전문가 여러분과 중소·중견기업 관련자 분께 깊이 감사드립니다.

일러두기

본 해설서는 W3C WebRTC 1.0 표준(WebRTC 1.0: Real-time Communication Between Browsers)문서 번역이 주 내용이다.

WebRTC개발자 중에서 표준문서를 읽어 본 개발자가 많지 않을 것이다. 이번 기회에 표준문서를 한번 읽어 봄으로서 WebRTC에 대해 좀 더 이해할 수 있는 기회가 되기를 기대한다.

해설서로서 번역문에 해설을 추가하였다면 더 좋았을 것이나, 그 영역이 방대하고, 역자마다 다르게 해설하거나 잘못이 있을 수 있어, 해설 없이 번역만 담았다. 원문 그대로 번역하다보니 매끄럽지 못한 부분이 있음에 양해를 구하는 바이다.

목차

목차

1. WebRTC 란?

WebRTC는 Web Real Time Communication의 약자이다. HTML5의 일부이며, 웹페이지에서 실시간으로 비디오, 오디오, 데이터 통신이 가능하도록 프로그래밍할 수 있다. 실시간통신을 위해 프로그램을 설치하거나 플러그인을 설치하지 않고도, 웹페이지에 접속하기만 하면, 영상, 오디오, 데이터 통신이 가능한 것이다.

W3C의 웹 표준에서 웹이 문서저장, 출판의 역할을 넘어 프로그래밍 가능한 플랫폼으로 진화하는데 실시간 미디어 통신에 대한 역할을 담당하고 있는 중요한 표준이다.

IETF(Internet Engineering Task Force, 국제 인터넷 표준화 기구) RTCWEB 워킹그룹에서 통신에 필요한 프로토콜 규격을 정의하고, W3C WebRTC 워킹그룹에서 로컬 미디어 장치 연동을 위한 API 규격을 정의한다.

2010년 구글이 주최하고 W3C, 모질라, 마이크로소프트, 인텔, 애플, 아이비엠, 에릭슨, 시스코, 로지텍, 오페라, 야후 등 여러 업체가 참석한 RTC Web 워크숍에서 웹에서 실시간 미디어 통신을 하기 위해 필요한 통신방법, 코덱, 암호화, 보안, API 수준 등의 내용들을 처음으로 논의했다.

WebRTC는 3가지 세트로 구성되어 있다고 할 수 있다.

1) 오픈소스 코덱(비디오, 오디오)을 포함한 미디어엔진 세트
   : 미디어엔진 내에는 실시간 미디어 송수신을 위한 처리 기술이 포함되어 있음
2) 통신 프로토콜 세트
3) 자바스크립트 API 세트

WebRTC의 대표적인 API는 아래와 같다.

1) getUserMedia : 미디어장치에 접근하여 캡처하기 위한 API
2) getDisplayMedia : 로컬 장치의 디스플레이에 접근하기 위한 API
3) MediaRecorder : 오디오, 비디오를 녹화하기 위한 API
4) RTCPeerConnection : 피어간에 스트림을 송수신하기 위한 연결 설정을 위한 API
5) RTCDataChannel : 피어간에 일반 임의 데이터(텍스트, 파일, 기타 데이터)를 송수신하기 위한 API
6) getStats : PeerConnection의 상태 정보에 접근하기 위한 API

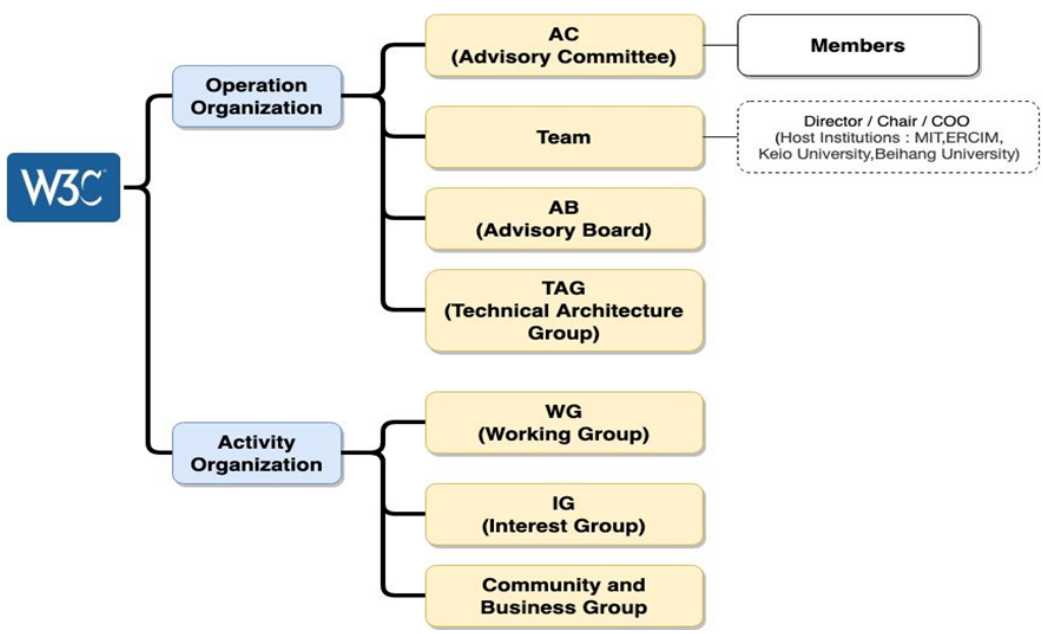
2. W3C 소개

월드와이드웹 컨소시엄(W3C)은 국제 커뮤니티이다. 영국 태생의 팀버너스리(Tim Berners-Lee)가 개발한 월드 와이드 웹의 표준을 함께 개발하기 위해 1994년 설립되었다.

팀버너스리는 이사로 참여하고 있고, CEO는 제프리 제프(Jeffrey Jaffe)이다.

스텝으로 참여하는 방법은 4개 운영팀을 통해 참여할 수 있다. 4개 운영팀은 MIT, ERCIM (유럽-유럽정보와수학연구컨소시엄), Keio 대학(일본), Beihang 대학(중국)이다.

1) W3C의 조직 구성도는 다음과 같다.



2) 조직별 주요역할

(1) 운영조직

- AC(Advisory Committee) : 자문위원회 TAG 위원 선출
- Team : W3C 호스트로서 운영을 담당하는 팀. 이사, 대표, 유무급 직원으로 구성되며 W3C 활동 지원/관리, 멤버 및 외부 교류
- AB(Advisory Board) : Team에게 다양한 분야에 대해 자문하는 주요 회원사로 구성된 자문위원회
- TAG(Technical Architecture Group) : 웹 기술에 대한 원칙을 문서화, W3C 내부 외부 기술간 조율

(2) 활동조직

- WG, IG, Community & Business Group : 기술별 표준문서화, 정기 모임

3) 회원 미팅 일정

- AC Meeting : 4월 or 5월, 웹표준의 전략, 향후 방향에 대해 논의하는 대면회의이다. 올해는 코로나19영향으로 온라인으로 진행되었다.
- TPAC Meeting(Technical Plenary and Advisory Committee) : 10월 or 11월,
- W3C 주요 기술 그룹과, 자문위원회 회원들이 모여 대면회의 및 토론을 진행한다. 올해는 코로나19영향으로 온라인으로 진행된다.
- 기타 비정기적인 온라인회의 수시 : 각 워킹그룹들은 비정기적으로 온라인 회의를 수시로 진행한다. 연중 진행된 온라인회의 안건들이 연말에 열리는 TPAC Meeting에서 재논의되거나 정리된다.

4) 전체 회원

- 445개 단체가 회원으로 등록되어 있다. (2020. 10월 현재)

5) 자문위원회(AB) 회원사(2020.10월)

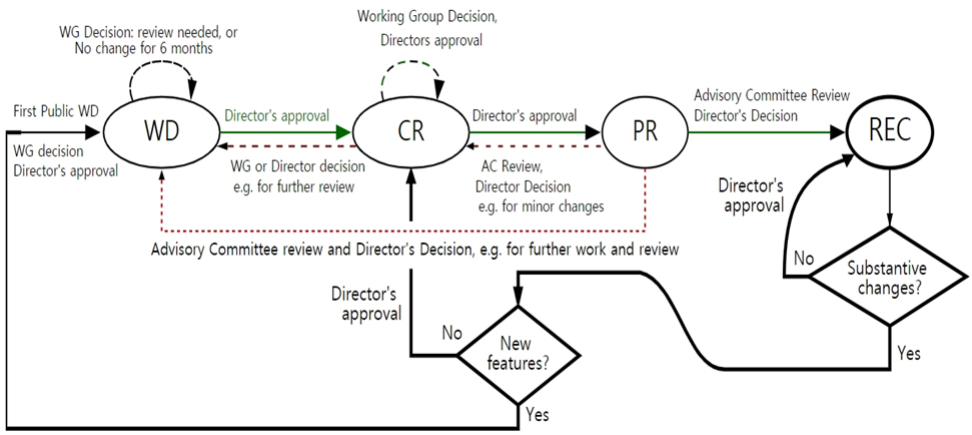
○ Mozila, Sony, Wiley, Apple, DAISY Consortium, Intel, TeraLogical, Google, Alibaba, W3C 초청 전문가

6) 회원자격 및 회비

○ 회원 자격은 모든 유형의 조직 및 개인
○ 연회비는 단체의 타입(위치, 규모, 영리/비영리)에 따라 차등 적용
○ 한국은 4개 호스트 팀 중 일본 Keio대학을 통해 가입하며, 타입에 따라 740만엔, 620만엔, 310만엔, 272만엔, 85만엔, 10만9천5백엔이다.

7) 표준화 절차

○ W3C의 표준화 절차는 "W3C Process Document"에 자세하게 명시되어 있다. (https://www.w3.org/Consortium/Process/)
○ 웹표준(Web Standard)이라하면, W3C가 게시하는 권고(Recommendation)문서를 의미한다.
○ 일반 절차
  • 특정 주제 논의/커뮤니티 단계 : 특정 주제에 관심 가진 회원은 W3C에 관심 사항을 제출하거나, 커뮤니티를 조직하여 사람들을 모아 워크숍을 진행한다.
  • 개발 제안 단계 : 특정 주제에 회원들의 충분한 관심이 있는 경우, 새로운 IC(Interest Group)나 WG(Working Group)의 활동 목적 페이지(헌장)를 제안하고, 회원들은 제안된 페이지를 검토한다.
  • 워킹 그룹은 주기적으로 기술리포트(Technical Report)인 사양 문서를 수정 및 업데이트 한다. 워킹 그룹이 구현 및 상호 운용성 경험을 보여줄 수 있는 요구 사항들이 포함되어 있는 문서이며, 문서 업데이트 프로세스가 끝나면, 자문위원회는 사양 문서를 확인하고, W3C 권장 사양으로 게시한다.
○ 표준화 절차 버전 : 관심 그룹이나, 커뮤니티그룹 등의 Note가 Working Draft가 되거나 바로 Working Draft 문서가 생성된다. Working Draft 문서가 Director의 인증을 거쳐 CR (Candidate Recommendation, 후보권고)로 상승하고 PR(Proposed Recommendation, 제안권고)단계에서 오류가 없으면 최종 REC(Recommendation, 권고) 단계가 된다.



8) WebRTC 워킹그룹

○ 웹브라우저에서 실시간 통신을 가능하게 하는 클라이언트 측 API 사양을 정의하는 것이다.

○ 그룹 참여자 수 : 24개 단체에 80명(2020. 10월)

○ 업무영역1(웹 브라우저간의 실시간 통신을 위한 클라이언트 측 기술 정의)

- 장치를 탐색하는 API(카메라, 마이크, 스피커)
- 장치를 캡처하는 API(카메라, 마이크, 화면과 같은 출력 장치)
- 미디어 스트림을 인코딩하고 이를 처리하기 위한 API
- 피어간에 데이터 전송을 위한 API
- 방화벽/NAT 통과를 포함한 피어간 직접 연결을 생성하기 위한 API
- 스트림 수신단에서 디코딩하고 이를 처리(에코 제거, 스트림 동기화 등)하기 위한 API
- 로컬 화면과 오디오 출력 장치를 통해 미디어 스트림의 사용자에게 전달에 관한 규격 정의

○ 업무영역2(웹 플랫폼의 다양한 컨텍스트에서 기능 사용 시 문제 해결)

- 기존 웹페이지
- 다양한 유형의 Web Workers
- 적절한 접근 컨트롤이 가능한 iframe
- 여러 페이지에 걸쳐있는 애플리케이션
- API 사용으로 영향을 주는 콘텐츠 보안정책 또는 기타 보안 메커니즘

3. WebRTC 활용 사례와 기술 스택

1) 활용 사례

WebRTC는 1:1 화상 통신을 넘어 다양한 분야에서 활용되고 있다.

- 화상채팅/소셜채팅
  Facebook Messenger, Google Duo, Snapchat, Houseparty, Azar, 스무디
- 화상회의/협업솔루션
  Whereby(구 Appearin), Google Meet / Hangout, Slack, MS Teams, Cisco
- Webex Meetings, Amazon Chime, Poly, Livesize, Gotomeeting, BlueJeans, Pexip, UberConference, uPrism.io Meetings, RemoteMeeting, Gooroomee
- 웨비나/행사
  demio, Bigmaker, webinarNinja, EventUs
- 파일공유/CDN
  Peer5, ShareDrop, FilePizza
- 게임
  Discord, Google Stadia
- 교육
  Minerva Project, BrainCert, Vedantu, dmm, WizIQ, Tutoring, CodeWings
- 컨택센터/콜센터
  Five9, Salesforce Service SOS, CafeX, VisualIVR
- 상담/컨설팅
  24Sessions, Wellbeing, 트로스트
- 방송
  MLB.com, Fox sports, MS Mixer, Comcast, RemoteMonster
- 원격 의료/헬스케어
  Dr. on demand, 2nd.MD, Revolve Kubi,
- PBX/전화연결
  Dialpad, Dialoga, Cloud PBX

- CCTV(IP 카메라)

  Flashphoner
- Door Bell/집 보안/홈오토메이션

  Ring (Amazon), vivint.SmartHome
- 로봇/웨어러블 디바이스

  doublerobotics, anybots, revolverrobotics
- 현장/고객 지원, AR/VR

  wavecell(8x8), Maxst,
- 드론/RC카

  Parrot
- 홈트레이닝

  PELOTON, yogaia
- 금융/은행

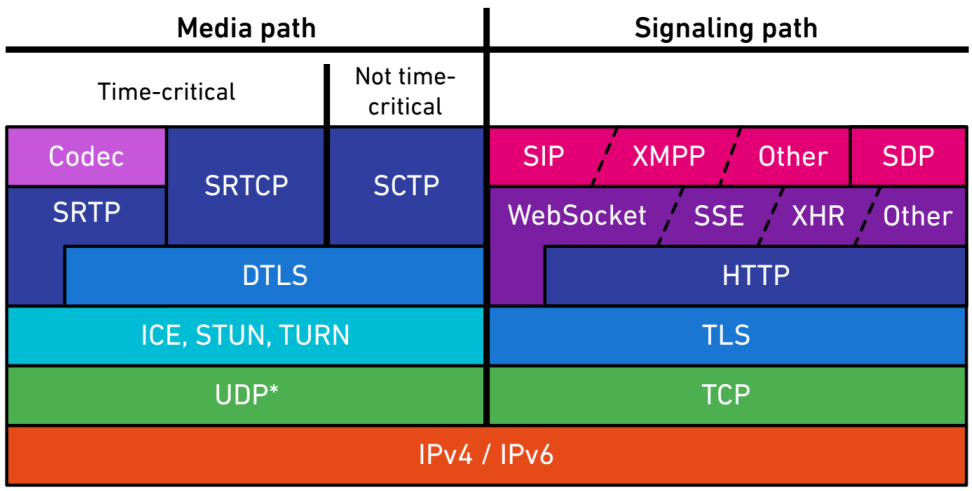
  Bank of America - ATM, Royal Bank of Scotland, symphony
- 면접/인터뷰

  cammio, codassium, VidCruiter
- 커넥티드카/자율주행차

  AT&T Drive, Quobis Sat2car
- Next Version New Use Cases (https://www.w3.org/TR/webrtc-nv-use-cases)

  Next Use Case 문서에는 WebRTC의 다음버전에서 지원해야할 사례가 명시되어 있다.
  - IoT
  - Funny Hats
  - Machine Learning
  - 가상현실 게임

2) 기술 스택



※ 출처 : Definition and Analysis of WebRTC Performance Parameters. December 2017 (Martin Meszaros, Franziska Trojahn)

위 이미지는 WebRTC Protocol Stack이다.

시그널링 메커니즘은 WebRTC API에 정의되어 있지 않다. 다양한 기존 프로토콜들을 사용할 수 있다. 예를 들면, 기존 VoIP 통신에서 사용하고 있는 SIP 프로토콜이나 메신저 프로토콜인 XMPP, 사물인터넷에서 사용하는 MQTT, WebSocket을 사용할 수도 있다.

4. WebRTC 1.0 표준문서 해설

1) WebRTC 1.0 표준문서 역사

2011년 5월 말에 구글이 처음으로 WebRTC 오픈소스 라이브러리를 공개하고, 같은 해 10월 W3C에서 WebRTC 1.0 Real-Time Communication Between Browsers 표준문서(Working Draft)가 처음 공개되었다.

WebRTC 표준이라 하면 WebRTC 1.0 Real-Time Communication Between Browsers(이하, "WebRTC 1.0 표준문서")를 지칭한다.

WebRTC 1.0 표준문서는 공식 게시 버전으로는 24번의 게시 버전이 있다.(2020년 10월 기준)

WebRTC 1.0 표준문서는 2017년 11월에 WD(Working Draft)버전에서 CR(Candidate Recommendation) 버전으로 업데이트되었고, 7년여의 시간이 지난 후 마침내 주요 사양이 완성된 것으로 평가 받는다.

WebRTC 워킹그룹이 협업하거나 관리하는 Web 표준은 16여 가지가 있으며, 그 수는 Recommen-dation 버전이 되기까지 더 늘어나거나 통합되어 줄어 들 수도 있다.

2) WebRTC 1.0 표준문서 내용

WebRTC 1.0 표준문서에는 다음의 API 규격과 내용이 포함되어 있다.

- P2P연결을 위한 RTCPeerConnection API
- 미디어 스트림을 송수신하는 데 필요한 RTP Media API
- 일반 데이터를 송수신 하는 데 필요한 Data API(RTCDataChannel)
- 전화기 키패드 Tone 신호를 보내는 데 사용되는 P2P DTMF(Dual Tone Multi Frequency, 이중 톤 다중 주파수) API(RTCDTMFSender)
- 통신 상태를 알수 있는 통계 모델(getStats)
- 기타, 에러 핸들링과 샘플

5. WebRTC 1.0 해설서

2020년 10월 현재 최신 WebRTC 1.0 표준문서 버전은 10월 15일 CR버전(https://www.w3.org/TR/2020/CRD-webrtc-20201015)이나, 본 공동번역 프로젝트가 시작된 시점인 9월 3일 CR버전(https://www.w3.org/TR/2020/CR-webrtc-20200903)을 한글 번역하였음을 참고해 주기 바란다.(최신 표준문서는 https://www.w3.org/TR/webrtc에서 확인할 수 있다.)

워킹그룹은 표준문서 완성을 마무리하기 위해 노력 중이다.

올해 안에는 PR(Proposed Recommendation)버전으로 업데이트될 것으로 예상된다.

번역문 참고 사항

본 번역문의 표준 버전은 2020년 9월 3일 CR버전이다.
공동 번역으로 번역자마다 해당 섹션에서 용어를 다르게 표시한 것 있다.
본 번역 문서는 오역이 있을 수 있으며, 필요시 영어 원문을 참조해야 한다.

WebRTC 1.0 : 브라우저간 실시간 통신(WebRTC 1.0: Real-time Communication Between Browsers)

W3C 후보 권고(Candidate Recommendation) 2020년 9월 3일

- 이 문서의 버전:
  https://www.w3.org/TR/2020/CR-webrtc-20200903/
- 발행된 최신 버전:
  https://www.w3.org/TR/webrtc/
- 편집자 최신 원고(Draft):
  https://w3c.github.io/webrtc-pc/
- 테스트 도구모음(Test suite):
  https://github.com/web-platform-tests/wpt/tree/master/webrtc/

○ 브라우저 구현 현황(Implementation report): https://wpt.fyi/webrtc

○ 이전 버전:
https://www.w3.org/TR/2020/CR-webrtc-20200825/

○ 편집자:
Cullen Jennings (Cisco)
Henrik Boström (Google)
Jan-Ivar Bruaroey (Mozilla)

○ 이전 편집자:
Adam Bergkvist (Ericsson) - Until 01 June 2018
Daniel C. Burnett (Invited Expert) - Until 01 June 2018
Anant Narayanan (Mozilla) - Until 01 November 2012
Bernard Aboba (Microsoft Corporation) - Until 01 March 2017
Taylor Brandstetter (Google) - Until 01 June 2018

○ 표준 참여하기:
GitHub w3c/webrtc-pc
File a bug
Commit history
Pull requests

○ 표준 정보 얻기:
Mailing list
IETF RTCWEB Working Group

이 규격서의 최초 작성자는 이안 힉슨(Ian Hickson, 구글)이었다. 저작권은 다음 저작권문을 따른다.

요약

이 표준문서는 실시간 프로토콜 세트가 구현된 다른 브라우저나 장치간에 미디어를 송수신 할 수 있는 WebIDL로 된 ECMAScript API 세트를 정의한다. 이 규격은 IETF RTCWEB 그룹이 개발한 프로토콜 규격과 로컬 미디어 장치에 접근하기 위한 API 규격이 결합하여 개발되고 있다.

이 문서의 현황

이 섹션에서는 이 문서 발행 시 현황을 설명한다. 다른 문서들이 이 버전의 문서를 대체할 수 있다. 현재 W3C 발행물들과 최신 버전의 기술문서는 https://www.w3.org/TR/에 있는 W3C technical reports index에서 확인할 수 있다.

이 API는 WHATWG에서 수행된 작업에 기반을 둔다.

이 규격서에는 기능상 완전하며 더 이상 실질적인 변경은 없을 것으로 예상된다. 이전 후보 권고안(Previous Candidate Recommendation) 이후 다음과 같은 변경사항이 규격에 반영되었다.

- WebRTC가 포트 스캔에 사용되는 것을 막기 위해 ICE 후보를 금지할 수 있게 허용함
- 다중 DTLS 인증 설정을 위한 지원은 더 이상 사용되지 않음
- voiceActivityFlag은 구현 부족으로 위험으로 표시됨
- PeerConnection을 종료할 때 상태표시가 명확해짐
- WebRTC 통계 최신 규정 반영됨

API의 브라우저 구현 호환성 보고페이지 작성을 위해 관련 테스트 도구가 활용될 것이다.

권장 권고안(Proposed Recommendation)으로 넘어가기 위해 적어도 두 개의 브라우저에서 기능이 구현되고 각 옵션 중 적어도 한 개의 브라우저에서 구현되어 시연이 가능할 것으로 예상한다. 한 개의 브라우저에만 구현된 필수 기능 기능은 개정된 후보 권고안에서 옵션으로 표시될 수 있다.

이 문서는 Web Real-Time Communications 워킹그룹에 의해 후보 권고안(Candidate Recommendation)으로 발행되었고, W3C 권고안이 되기 위한 것이다.

이 규격에 대한 논의는 GitHub에 있는 이슈를 통해 진행된다. 다른 방법은 우리의 메일링리스트 [public-webrtc@w3.org (아카이브)]로 의견을 보낼 수 있다.

W3C는 이 문서가 안정화 버전이 될 것임을 암시하고, 개발자 커뮤니티에 의한 구현을 독려하기 위해 후보 권고안을 발행한다. 이 후보 권고안은 빠르면 2020년 9월 24일 이전에 제안 권고안(Proposed Recommendation)으로 승격될 수 있다. (한글 번역 역자 주 - 이 문장 때문에 역자는 9월 24일에는 PR버전으로 승격될 것을 기대하였으나, 승격되지 않았고, 다음 버전에서는 이 문장이 사라졌다.)

워킹 그룹의 브라우저 구현 테스트 사이트를 확인하기를 바란다.

후보 권고안으로 출판은 W3C 회원들의 승인을 의미하는 것은 아니다. 이 문서는 업데이트될 수 있고, 다른 문서에 의해 언제든지 대체 되거나 폐기될 수 있다. 이 문서를 인용 시에는 진행 중인 작업으로만 인용되어야 한다.

이 문서는 W3C 특허 정책 하에 운영되는 그룹에 의해 작성되었다. W3C는 다른 그룹과 연결된 특허 공개 리스트를 유지하며, 그 페이지는 특허 공개 지침도 포함한다. 특허에 대한 핵심 주장을 가지고 있는 사람은 W3C 특허 정책 6장에 따라 그 정보를 공개해야 한다.

이 문서는 W3C 표준 진행 문서(2019년 3월 1일 버전)에 의한 진행 적용을 받는다.

1) 소개

이 절은 비표준 부분이다.

이 규격은 HTML 안에서 P2P 통신과 화상 회의를 하기 위한 여러 가지 측면을 다룬다.

- ICE, STUN 및 TURN과 같은 NAT 통과 기술을 사용하여 원격 피어에 연결하기
- 로컬에서 생성된 트랙을 원격 피어로 전송하고 원격 피어에서 트랙을 수신하기
- 임의의 데이터를 원격 피어에 직접 보내기

이 문서는 이러한 기능에 사용되는 API를 정의한다. 이 사양은 IETF RTCWEB group 그룹에서 개발한 프로토콜 규격 및 WebRTC 작업 그룹에서 개발한 로컬 미디어 장치 [GETUSERMEDIA]에 대한 액세스 권한을 얻기 위한 API 규격과 함께 개발되고 있다. 이 시스템의 개요는 [RTCWEB-OVERVIEW] 및 [RTCWEB-SECURITY]에서 확인할 수 있다.

2) 준수(요구사항과 권고사항)

이 규격의 모든 저작 지침, 다이어그램, 예제 및 노트의 내용은 비표준이다. 그 외 다른 내용은 표준이다.

문서 내에 MAY, MUST, MUST NOT, SHOULD가 모두 대문자로 표시된 경우 BCP 14 [RFC2119] [RFC8174]에 설명대로 해석한다.

이 규격은 단일 제품(준수 기준을 충족한 인터페이스를 구현하는 사용자 에이전트)에 적용되는 준수할 기준을 정의한다.

알고리즘이나 규격으로 설명된 준수 요구사항은 최종결과가 같은 한 어떤 식으로든 구현할 수 있다. (특히, 본 규격에 정의 된 알고리즘은 따라하기 쉽도록 의도되었고 성능을 위해 의도된 것이 아니다.)

ECMAScript를 사용하여 이 규격에 정의된 API를 구현하는 구현사항들은 이 규격이 해당 사양과 용어를 사용하므로 Web IDL 규격 [WEBIDL]에 정의된 ECMAScript Bindings와 일치하는 방식으로 구현해야 한다.

3) 용어

이벤트 핸들러에 사용되는 콜백을 나타내는 이벤트 핸들러(Event Handler) 인터페이스는 [HTML]에 정의되어 있다.

작업 대기열(queue a task)과 네트워크 작업 소스(networking task source) 개념은 [HTML]에 정의되어 있다.

이벤트 발생(fire an event) 개념은 [DOM]에 정의되어 있다.

이벤트, 이벤트 핸들러와 이벤트 핸들러 이벤트 타입 용어는 [HTML]에 정의되어 있다.

Performance.timeOrigin와 Performance.now()는 [hr-time]에 정의되어 있다.

직렬화 가능 객체(serializable objects), 직렬화 단계(serialization steps)와 역직렬화 단계(deserialization steps) 용어들은 [HTML]에 정의되어 있다.

MediaStream, MediaStreamTrack과 MediaStreamConstraints 용어들은 [GETUSERMEDIA]에 정의되어 있다. 참고로 MediaStream은 이 문서의 § 9.2 MediaStream에서 확장된 것이고, MediaStreamTrack은 이 문서의 § 9.3 MediaStreamTrack에서 확장된 것이다.

Blob용어는 [FILEAPI]에 정의되어 있다.

미디어 설명(media description)은 [RFC4566]에 정의되어 있다.

미디어 전송(media transport)은 [RFC7656]에 정의되어 있다.

생성(generation)은 [TRICKLE-ICE] Section 2에 정의되어 있다.

통계객체(stats object)와 모니터링 대상 객체(monitored object)는 [WEBRTC-STATS]에 정의되어 있다.

예외처리를 언급할 때 예외 던짐(throw)과 예외 생성(created)라는 용어는 [WEBIDL]에 정의되어 있다.

VoidFunction 콜백은 [WEBIDL]에 정의되어 있다.

“throw”라는 용어는 [INFRA]에 명시된 대로 사용된다 : “throw”는 현재 처리 단계를 종료한다.

자바스크립트 Promises의 맥락으로 사용되는 이행(fulfilled), 거부(rejected), 해결(resolved), 대기(pending) 및 처리(settled)라는 용어는 [ECMASCRIPT-6.0]에 정의되어 있다.

번들(bundle), 번들 전용(bundle-only) 및 번들 정책(bundle-policy) 용어들은 [JSEP]에 정의되어 있다.

알고리즘식별자(AlgorithmIdentifier)는 [WebCryptoAPI]에 정의되어 있다.

참고

[API-DESIGN-PRINCIPLES]정의된 실행완료(run-to-completion) 및 데이터 경합 없음(no-data-races)의 원칙을 포함한 자바스크립트 API 기본원칙이 적용된다.

즉, 작업이 실행되는 동안 외부 이벤트는 자바스크립트 애플리케이션에 표시되는 항목에 영향을 주지 않는다. 예를 들어, 데이터 채널에 버퍼링되는 데이터의 양은 Javascript가 실행되는 동안 “send” 호출로 인해 증가하고 전송되는 패킷으로 인한 감소는 작업 점검점 이후에 표시된다.

애플리케이션에 제공된 값 세트가 일관 적인지 확인하는 것은 user agent의 책임이다. 예를 들어 getContributingSources () (동기식)는 동시에 측정 된 모든 소스에 대한 값을 반환한다.

4) P2P 연결(Peer-to-peer connections)

(1) 소개

RTCPeerConnection 인스턴스는 애플리케이션이 다른 브라우저에서 다른 RTCPeerConnection 인스턴스나, 필요한 프로토콜을 구현하는 다른 엔드포인트와의 P2P 통신을 설정할 수 있다. 피어 연결 통신은 시그널링 채널을 통해 컨트롤 메시지 교환(시그널링 프로토콜 이라고 함)을 하고 조정하게 되는데, 그 수단을 규정하고 있지는 않다. 일반적으로 서버를 통해 페이지의 스크립트에 의해 제공되는 프로토콜을 사용한다. 예: Web Socket 또는 XMLHttpRequest [xhr].

(2) 설정(Configuration)

① RTCConfiguration 사전 (RTCConfiguration Dictionary)

RTCConfiguration은 RTCPeerConnection을 통해 설정된 P2P 통신이 설정되거나 재설정된 방법을 구성하는 매개 변수 집합을 정의한다.

WebIDL

```
dictionary RTCConfiguration {
    sequence<RTCIceServer> iceServers;
        RTCIceTransportPolicy iceTransportPolicy;
        RTCBundlePolicy bundlePolicy;
        RTCRtcpMuxPolicy rtcpMuxPolicy;
        sequence<RTCCertificate> certificates;
        [EnforceRange] octet iceCandidatePoolSize = 0;
};
```

RTCConfiguration 사전 멤버(Dictionary RTCConfiguration Members)

iceServers(<RTCIceServer>의 시퀀스 타입)

STUN과 TURN 서버 같은 ICE에서 사용할 서버들의 개체 배열.

iceTransportPolicy (RTCIceTransportPolicy 타입)

사용할 수 있는ICE 에이전트 후보를 나타냄.

bundlePolicy(RTCBundlePolicy 타입)

ICE 후보를 수집할 때 사용할 미디어 번들 정책을 나타냄.

rtcpMuxPolicy(RTCRtcpMuxPolicy 타입)

ICE 후보를 수집할 때 사용할 rtcp-mux 정책을 나타냄.

certificates(<RTCCertificate> 시퀀스 타입).

RTCPeerConnection이 인증하는 데 사용하는 인증서 집합.
이 매개 변수의 유효값은 generateCertificate() 함수 호출을 통해 생성.

주어진 DTLS 연결은 하나의 인증서만 사용하지만, 이 속성을 사용하면 호출자가 다른 알고리즘을 지원하는 여러 인증서를 제공 할 수 있다. 최종 인증서는 허용되는 인증서를 설정하는 DTLS 핸드셰이크를 기반으로 선택된다. RTCPeerConnection 구현은 주어진 연결에 사용되는 인증서 중 하나를 선택한다; 인증서 선택 방법은 이 규격의 범위를 벗어난다.

참고

기존 구현은 제공된 첫 번째 인증서만 사용하고 나머지는 무시한다.

이 값이 없으면 각 RTCPeerConnection 인스턴스를 위한 기본 인증서 집합이 생성된다.

이 옵션은 응용프로그램이 키 연속성(key continuity) 생성을 가능하게 한다. RTCCertificate는 [INDEXEDDB]에 유지되고 재사용 할 수 있다. 또한, 지속성 및 재사용은 키 생성 비용을 절감한다.

이 구성 옵션의 값은 처음 선택한 후에는 변경할 수 없다.

iceCandidatePoolSize(octet 타입, 기본값 0)

프리페치(prefetched)된 ICE 풀의 크기([JSEP] (섹션 3.5.4 및 섹션4.1.1)에 정의된)

② RTCIceCredentialType 열거형(RTCIceCredentialType Enum)

WebIDL

```
enum RTCIceCredentialType {
    "password"
};
```

열거형 설명(Enumeration description)	
password	인증 정보는 [RFC5389] 섹션 10.2에 기술된 바와 같이 장기 인증 사용자명과 비밀번호이다.

③ RTCIceServer 사전(RTCIceServer Dictionary)

RTCCIceServer 사전은 ICE 에이전트가 피어와의 연결을 설정하는데 사용할 수 있는 STUN 및 TURN 서버를 설명하는 데 사용된다.

WebIDL

```
dictionary RTCIceServer {
    required (DOMString or sequence<DOMString>) urls;
        DOMString username;
        DOMString credential;
        RTCIceCredentialType credentialType = "password";
};
```

RTCIceServer 사전 멤버

Urls(DOMString 또는 <DOMString>시퀀스 타입, 필수)

STUN 또는 TURN URI.([RFC7064] 및 [RFC7065] 또는 기타 URI 유형에 정의된)

username(DOMString 타입)

만약 이 RTCIceServer 객체가 TURN 서버이며 credentialType이 “password"인 경우, 이 속성은 (TURN 서버와 함께 사용할)사용자명을 지정한다.

credential(DOMString 타입)

만약, 이 RTCIceServer 객체가 TURN 서버인 경우, 이 속성은 TURN 서버와 함께 사용할 자격 증명을 지정한다.

If credentialType is “password”, credential represents a long-term authentication password, as described in [RFC5389], Section 10.2.
만약, credentialType이 “password”인 경우, credential은 [RFC5389] 섹션 10.2에 설명된 장기 인증 비밀번호를 의미한다.

참고

credentialType의 추가적인 값 지원을 위해, credential은 향후 union 형태로 발전할 수 있다.

credentialType(RTCIceCredentialType 타입, 기본값 "password")

만약, 이 RTCIceServer 객체가 TURN 서버인 경우, 이 속성은 TURN 서버의 권한 요청에 대한 자격 증명 방법을 지정한다.

RTCIceServer 객체 배열 예제:

Example 1

```
예제 1
[
    {urls: 'stun:stun1.example.net'},
        {urls: ['turns:turn.example.org', 'turn:turn.example.net'],
        username: 'user',
        credential: 'myPassword',
        credentialType: 'password'},
];
```

④ RTCIceTransportPolicy 열거형

[JSEP](섹션 4.1.1)에 설명된 바와 같이, RTCConfiguration의 iceTransportPolicy 멤버가 지정되면 브라우저가 허용 된 후보의 애플리케이션에 대해 표면화하는 데 사용하는 ICE 후보 정책[JSEP](섹션 3.5.3.)을 정의한다. 오직 이 후보자들만 연결 검사에 사용된다.

WebIDL

```
enum RTCIceTransportPolicy {
    "relay",
    "all"
};
```

열거형	설명(비표준)
relay	ICE 에이전트는 TURN 서버를 통과하는 후보와 같은 미디어 릴레이 후보만 사용한다. **참고** 이는 원격 엔드포인트가 특정 사용 사례에서 필요할 수 있는 사용자의 IP 주소를 학습하는 것을 방지하는 데 사용할 수 있다. 예를 들어, "통화" 기반 애플리케이션에서, 애플리케이션은 수신자가 동의할 때까지 알 수 없는 발신자가 어떤 식으로든 수신자의 IP 주소를 학습하지 못하도록 할 수 있다.
all	ICE 에이전트는 이 값이 지정되면 모든 유형의 후보를 사용할 수 있다. **참고** 구현은 RTCIceCAndidate.address의 설명에 명시된 대로 애플리케이션에 노출된 IP 주소를 제한하기 위해 자체 후보 필터링 정책을 계속 사용할 수 있다.

⑤ RTCBundlePolicy 열거형

[JSEP](섹션 4.1.1.)에 설명된 바와 같이, 번들 정책은 원격 엔드포인트가 번들을 인식하지 않는 경우 협상되는 미디어 트랙과 수집되는 ICE 후보에 영향을 준다. 원격 엔드포인트가 번들을 인식하는 경우, 모든 미디어 트랙과 데이터 채널이 동일한 전송에 번들로 제공된다.

WebIDL

```
enum RTCBundlePolicy {
    "balanced",
    "max-compat",
    "max-bundle"
};
```

열거형	설명(비표준)
balanced	사용 중인 각 미디어 유형(오디오, 비디오 및 데이터)에 대한 ICE 후보를 수집한다. 원격 엔드포인트가 번들을 인식하지 못하는 경우, 별도의 전송에서 하나의 오디오 및 비디오 트랙만 협상한다.
max-compat	각 트랙에 대한 ICE 후보를 수집한다. 원격 엔드포인트가 번들을 인식하지 못하는 경우, 별도의 전송에서 모든 미디어 트랙을 협상한다.
max-bundle	하나의 트랙에 대해서만 ICE 후보를 수집한다. 원격 엔드포인트가 번들을 인식하지 못하는 경우, 하나의 미디어 트랙만 협상한다.

⑥ RTCRtcpMuxPolicy 열거형

[JSEP](섹션 4.1.1.)에 설명된 바와 같이, RTCRtcpMuxPolicy는 다중화되지 않은 RTCP를 지원하기 위해 수집된 ICE 후보에 영향을 준다. 이 규격에 정의된 유일한 값은 "require"이다.

WebIDL

```
enum RTCRtcpMuxPolicy {
    "require"
};
```

열거형	설명(비표준)
require	RTP 후보에서 RTP 및 다중 RTCP에 대해서만 ICE 후보를 수집한다. 원격 엔드포인트가 rtcp-mux를 사용할 수 없는 경우, 세션 협상은 실패한다.

⑦ 제안/응답(Offer/Answer) 옵션

이 사전은 제안/응답 생성 프로세스를 제어하는 데 사용할 수 있는 옵션을 설명한다.

WebIDL

```
dictionary RTCOfferAnswerOptions {};
```

RTCOfferAnswerOptions 사전 멤버

WebIDL

```
dictionary RTCOfferOptions : RTCOfferAnswerOptions {
    boolean iceRestart = false;
};
```

RTCOfferOptions 사전 멤버

iceRestart(boolean 타입, 기본 값 false)

이 사전 멤버의 값이 true이거나, 관련 RTCPeerConnection 객체의 [[LocallceCredentials ToReplace]] 슬롯이 비어 있지 않은 경우, 생성된 설명은 현재 자격 증명과 다른 ICE 자격 증명을 갖게 된다(currentLocalDescription 속성의 SDP에 표시됨). 생성된 설명을 적용하면 [ICE]의 섹션 9.1.1.1에 설명된 대로 ICE가 다시 시작된다.

이 사전 멤버의 값이 false이고 관련 RTCPeerConnection 객체의 [[LocalIceCredentials ToReplace]] 슬롯이 비어 있고, currentLocalDescription 속성에 유효한 ICE 자격 증명이 있는 경우, 생성된 설명은 currentLocalDescription의 속성에 현재 값과 동일한 ICE 자격 증명을 갖는다.

참고

iceConnectionState가 "failed"로 전환되면 ICE 재시작을 권장한다. 애플리케이션은 추가적으로 iceConnectionState가 "disconnected"으로 전환되는 것을 수신한 다음 다른 정보 소스(getStats를 사용하여 향후 몇 초 동안 송신 또는 수신한 바이트 수가 증가하는지 측정과 같은 경우)를 사용하여 ICE 재시작의 권장 여부를 결정할 수 있다.

RTCAnswerOptions 사전은 "answer" 타입의 세션 설명에 특정한 옵션을 설명한다(이 버전의 규격에는 없음).

WebIDL

```
dictionary RTCAnswerOptions : RTCOfferAnswerOptions {};
```

(3) 상태 정의(State Definitions)

① RTCSignalingState 열거형

WebIDL

```
enum RTCSignalingState {
    "stable",
    "have-local-offer",
    "have-remote-offer",
    "have-local-pranswer",
    "have-remote-pranswer",
    "closed"
};
```

열거형	설명
stable	진행 중인 제안/답변 교환이 없다. 또한 로컬 및 원격 설명이 비어 있는 초기 상태이다.
have-local-offer	“offer” 타입의 로컬 설명이 성공적으로 적용되었다.
have-remote-offer	“offer” 타입의 원격 설명이 성공적으로 적용되었다.
have-local-pranswer	“offer” 타입의 원격 설명이 성공적으로 적용되었으며 “pranswer” 타입의 로컬 설명이 성공적으로 적용되었다.
have-remote-pranswer	“offer” 타입의 로컬 설명이 성공적으로 적용되었으며 “pranswer” 타입의 원격 설명이 성공적으로 적용되었다.
closed	RTCPeerConnection이 닫혔다; [[IsClosed]] 슬롯이 true이다.

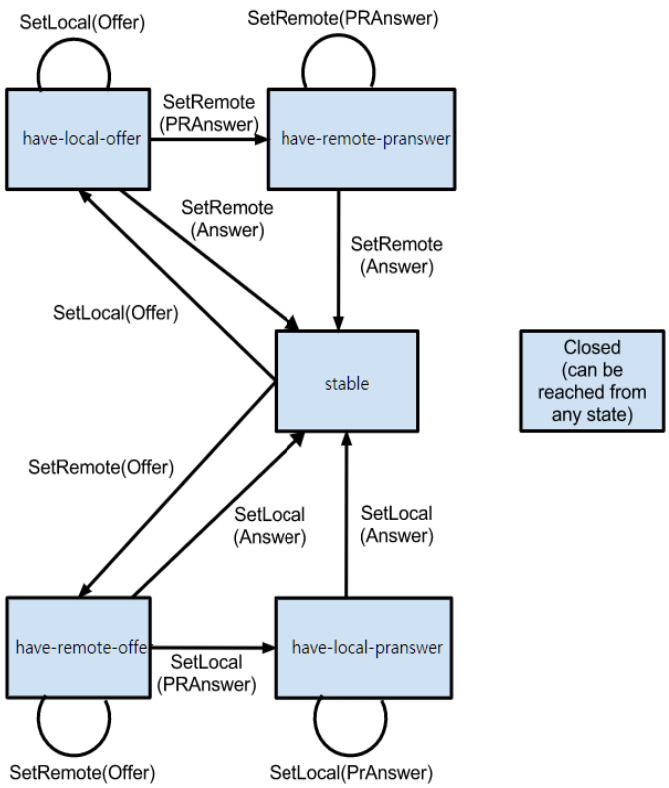


그림 1. 비표준 시그널링(signaling) 상태 이행과정 다이어그램. 메소드 호출 약어

이행 과정의 예는 다음과 같다:

발신자 이행 과정 :

- new RTCPeerConnection(): "stable"
- setLocalDescription(offer): "have-local-offer"
- setRemoteDescription(pranswer): "have-remote-pranswer"
- setRemoteDescription(answer): "stable"

수신자 이행 과정 :

- new RTCPeerConnection(): "stable"
- setRemoteDescription(offer): "have-remote-offer"
- setLocalDescription(pranswer): "have-local-pranswer"
- setLocalDescription(answer): "stable"

② RTCIceGatheringState 열거형

WebIDL

```
enum RTCIceGatheringState {
    "new",
    "gathering",
    "complete"
};
```

열거형	설명
new	RTCIceTransport는 “new” 수집 상태에 있고, 어떤 전송도 “gathering” 상태에 있지 않거나 전송이 없다.
gathering	RTCIceTransport가 “gathering” 상태에 있다.
complete	적어도 하나 이상의 RTCIceTransport가 존재하고, 모든 RTCIceTransport가 “complete” 수집 상태에 있다.

③ RTCPeerConnectionState 열거형

WebIDL

```
enum RTCPeerConnectionState {
    "closed",
    "failed",
    "disconnected",
    "new",
    "connecting",
    "connected"
};
```

열거형	설명
closed	RTCPeerConnection 객체의 [[IsClosed]] 슬롯은 true이다.
failed	이전 상태가 적용되지 않고 RTCIceTransport가 “failed” 상태에 있거나 RTCDtlsTransport가 “failed” 상태에 있다.
disconnected	이전 상태가 적용되지 않으며 RTCIceTransport가 “disconnected” 상태에 있다.
new	이전 상태가 적용되지 않으며 모든 RTCIceTransport는 “new”나 “closed” 상태에 있고, 모든 RTCDtlsTransport는 “new” 혹은 “closed” 상태이거나, 전송이 없다.
connecting	이전 상태가 적용되지 않으며 RTCIceTransport가 “checking” 상태이거나 RTCDtlsTransport는 “connecting” 상태에 있다.
connected	이전 상태가 적용되지 않으며 모든 RTCIceTransport가 “connected”, “completed” 또는 “closed” 상태이고, 모든 RTCDtlsTransport가 “connected” 또는 “closed” 상태에 있다.

④ RTCIceConnectionState 열거형

WebIDL

```
enum RTCIceConnectionState {
    "closed",
    "failed",
    "disconnected",
    "new",
    "checking",
    "completed",
    "connected"
};
```

열거형	설명
closed	RTCPeerConnection 객체의 [[IsClosed]] 슬롯은 true이다.
failed	이전 상태는 적용되지 않으며 RTCIceTransport는 “failed” 상태에 있다.
disconnected	이전 상태가 적용되지 않으며 RTCIceTransport는 “disconnected” 상태에 있다.
new	이전 상태가 적용되지 않으며 모든 RTCIceTransport가 “new” 혹은 “closed” 상태이거나, 전송이 없다.
checking	이전 상태가 적용되지 않으며 RTCIceTransport는 “new” 이거나 “checking” 상태이다.
completed	이전 상태가 적용되지 않으며 모든 RTCIceTransport는 “completed” 이거나 “closed” 상태이다.
connected	이전 상태가 적용되지 않으며 모든 RTCIceTransport는 “connected”, "completed" 이거나 “closed” 상태이다.

참고로 RTCIceTransport가 시그널링(signaling)(예: RTCP mux 혹은 번들링)의 결과로 폐기되거나 시그널링(signaling)(예: 새로운 미디어 설명 추가)의 결과로 생성된 경우, 한 상태에서 다른 상태로 바로 진행될 수 있다.

(4) RTCPeerConnection 인터페이스

[JSEP] 규격은 전체적으로 RTCPeerConnection 작동 방법에 대한 세부적인 사항을 설명한다. [JSEP]의 특정 하위 섹션에 대한 참조가 적절히 제공된다.

① 작동(Operation)

new RTCPeerConnection(configuration)을 호출하면 RTCPeerConnection 객체가 생성된다.

configuration.iceServers는 ICE에서 사용하는 서버를 찾아 액세스하는 데 사용되는 정보를 담고 있다. 애플리케이션은 각 유형의 여러 서버를 공급할 수 있으며, 어떤 TURN 서버는 서버 재귀 후보 수집의 목적 TURN 서버로 사용될 수 있다.

RTCPeerConnection 객체는 시그널링(signaling) 상태, 연결 상태, ICE 수집 상태 및 ICE 연결 상태를 갖는다. 객체가 생성될 때 초기화된다.

RTCPeerConnection의 ICE 프로토콜 구현은 ICE 에이전트에 의해 표현된다[ICE]. 특정 RTCPeerConnection 메소드는 ICE 에이전트와의 상호작용, 즉 addIceCandidate, setConfiguration, setLocalDescription, setRemoteDescription 및 close가 포함된다. 이러한 상호작용은 본 문서와 [JSEP]의 관련 섹션에 설명되어 있다. ICE 에이전트는 또한 5.6 RTCIceTransport 인터페이스에 설명된 대로 RTCIceTransport의 내부 표현 상태가 변경될 때 사용자 에이전트에 표시를 제공한다.

이 섹션에 나열된 작업에 대한 작업 소스는 네트워킹 작업 소스이다.

참고

SDP 협상 상태는 시그널 상태와 내부 변수 [[CurrentLocalDescription]], [[CurrentRemoteDescription]], [[PendingLocalDescription]] 및 [[PendingRemoteDescription]]로 표시된다. 이는 setLocalDescription 및 setRemoteDescription 작업 내에서만 설정되며, addIceCandidate 작업 및 표면 후보 프로시저에 의해서 수정된다. 각각의 경우 프로시저가 이벤트를 발생하거나 콜백을 호출하기 전에 5개의 변수 모두에 대한 모든 수정이 완료되므로, 수정 사항은 단일 시점에 표시된다.

언로드 문서 정리 단계(unloading document cleanup steps) 중 하나로 다음 단계를 실행해야 한다:

1. window를 document의 관련 전역 객체(relevant global object)로 설정한다.
2. 관련 전역 객체가 window인 각 RTCPeerConnection 객체의 connection에 대해, connection과 값이 true인 연결을 닫는다.

ⓐ 생성자(Constructor)

RTCPeerConnection.constructor()가 호출되면, 사용자 에이전트는 다음 단계를 반드시(MUST) 실행해야 한다:

1. 만약 여기에 지정되지 않은 이유로 아래 열거된 단계 중 하나라도 실패하면 message 속성에 적절하게 설명된 UnknownError를 예외 던짐(throw)발생시킨다.
2. connection을 새로 생성된 RTCPeerConnection 객체로 만든다.
3. Connection을 [[DocumentOring]] 내부 슬롯에 있고, 현재 설정을 개체의 원본으로 초기화한다.
4. configuration을 메소드의 첫 번째 인수로 지정한다.
5. configuration의 certificates 값이 비어 있지 않으면, 인증서의 각 certificate에 대해 다음 단계를 실행한다:
   1) certificate.expires의 값이 현재 시간보다 늦으면, InvalidAccessError를 발생(throw) 시킨다.
   2) certificate.[[Origin]]이 connection.[[DocumentOrigin]]과 동일한 출처가 아니면, InvalidAccessError를 발생(throw)시킨다.
   3) certificate를 저장한다.
6. 그렇지 않으면, RTCPeerConnection 인스턴스로 하나 이상의 새로운 RTCCertificate 인스턴스를 생성하고 저장한다. 이것은 비동기적으로 발생할 수(MAY) 있으며 certificates의 값은 다음 단계에서 undefined로 유지된다. [RTCWEB-SECURITY]의 섹션 4.3.2.3의 언급과 같이 WebRTC는 공개 키 인프라(PKI) 인증서보다는 자체 서명된 인증서를 사용하므로 만료 확인은 키가 무기한 사용되지 않고 추가 인정서 확인이 필요하지 않은지 체크한다.
7. connection의 ICE 에이전트를 초기화한다.

8. configuration.iceTransportPolicy의 값이 undefined이면, "all"로 설정한다.
9. configuration.bundlePolicy의 값이 undefined이면, 로 "balanced"로 설정한다.
10. configuration.rtcpMuxPolicy의 값이 undefined이면, "require"로 설정한다.
11. connection에 [[Configuration]] 내부 슬롯을 설정한다. configuration에서 지정한 구성을 설정한다.
12. connection에 [[IsClosed]] 내부 슬롯을 설정하고, fals로 초기화한다.
13. connection에 [[NegotiationNeeded]] 내부 슬롯을 설정하고, false로 초기화한다.
14. connection에 [[SctpTransport]] 내부 슬롯을 설정하고, null로 초기화한다.
15. connection에 연산 체인을 나타내는 [[Operations]] 내부 슬롯을 설정하고, 빈 리스트로 초기화한다.
16. connection에 [[UpdateNegotiationNeededFlagOnEmptyChain]] 내부 슬롯을 설정하고, false로 초기화한다.
17. connection에 [[LastCreatedOffer]] 내부 슬롯을 설정하고, ""로 초기화한다.
18. connection에 [[LastCreatedAnswer]] 내부 슬롯을 설정하고, ""로 초기화한다.
19. connection에 [[EarlyCandidates]] 내부 슬롯을 설정하고, 빈 리스트로 초기화한다.
20. connection의 시그널링(signaling) 상태를 "stable"로 설정한다.
21. connection의 ICE 연결 상태를 "new"로 설정한다.
22. connection의 수집 상태를 "new"로 설정한다.
23. connection의 연결 상태를 "new"로 설정한다.
24. connection에 [[PendingLocalDescription]] 내부 슬롯을 설정하고, null로 초기화한다.
25. connection에 [[CurrentLocalDescription]] 내부 슬롯을 설정하고, null로 초기화한다.
26. connection에 [[PendingRemoteDescription]] 내부 슬롯을 설정하고, null로 초기화한다.
27. connection에 [[CurrentRemoteDescription]] 내부 슬롯을 설정하고, null로 초기화한다.
28. connection에 [[LocalIceCredentialsToReplace]] 내부 슬롯을 설정하고, 빈(empty set)으로 초기화한다.
29. connection을 리턴한다.

ⓑ 비동기 작업 연결(Chain an asynchronous operation)

RTCPeerConnection 객체에는 체인에서 하나의 비동기 작업만 동시에 실행되도록 하는 작업 체인 [[Operations]]이 있다. 이전 호출의 리턴된 promise가 여전히 해결되지 않은 상태에서 후속 호출이 이루어지면, 이들은 체인에 추가되고 이전 호출이 모두 실행을 완료하고 promise가 해결되었을 때 실행된다.

RTCPeerConnection 객체의 작업 체인에 작업을 연결(to chain an operation)하려면 다음 단계들을 실행한다:

1. Connection을 RTCPeerConnection 객체로 지정한다.
2. connection.[[IsClosed]]가 true이면, 새로 생성된 InvalidStateError와 함께 거부된 promise를 반환한다.
3. Operation은 chained.operation 형식이 되게 한다.
4. p를 새로운 promise가 되게 한다.
5. [[Operations]]에 operation을 추가한다.
6. [[Operations]]의 길이가 정확히 1이면, operation을 실행한다.
7. operation에 의해 반환된 promise를 이행하거나 거부할 경우, 다음 단계를 실행한다:
   1) connection.[[IsClosed]]가 true이면, 이 단계를 중단한다.
   2) operation에서 반환된 promise가 값으로 이행된 경우, 해당 값으로 p를 이행한다.
   3) operation에서 반환된 promise가 값으로 거부 된 경우, 해당 값으로 p를 거부한다.
   4) p의 이행 또는 거부 시 다음 단계를 실행한다:
      (1) connection.[[IsClosed]]가 true이면, 이 단계를 중단한다.
      (2) [[Operations]]의 첫 번째 요소를 제거한다.
      (3) [[Operations]]가 비어 있지 않으면, [[Operations]]의 첫 번째 요소가 나타내는 작업을 실행하고, 이 단계를 중단한다.
      (4) connection.[[UpdateNegotiationNeededFlagOnEmptyChain]]가 false이면, 이 단계를 중단한다.
      (5) connection.[[UpdateNegotiationNeededFlagOnEmptyChain]]을 false로 설정한다.
      (6) connection에 대한 협상 필요 플래그(negotiatin-needed flag)를 업데이트 한다.
8. p를 리턴한다.

ⓒ 연결 상태 업데이트(Update the connection state)

RTCPeerConnection 객체는 집계된 연결 상태를 가진다. RTCDtlsTransport의 상태가 변경되거나 [[IsClosed]] 슬롯이 true가 될 때마다 사용자 에이전트는 다음 단계를 실행하는 작업을 대기열(queue)에 반드시 추가하여 연결 상태를 업데이트 해야한다(MUST):

1. connection을 이 RTCPeerConnection 객체로 지정한다.
2. newState를 RTCPeerConnectionState 열겨된 새 상태값으로 도출된 값으로 지정한다.
3. connection의 연결 상태가 newState와 같으면 이 단계를 중단한다.
4. connection의 연결 상태를 newState로 지정한다.
5. connection에서 connectionstatechange이라는 이벤트를 발생(Fire and event)시킨다.

ⓓ ICE 수집 상태 업데이트 (Update the ICE gathering state)

RTCPeerConnection 인스턴스 connection의 ICE 수집 상태를 업데이트하려면, 사용자 에이전트는 다음 단계를 실행하는 작업(task)을 대기열(queue)에 반드시(MUST) 넣어야 한다:

1. connection.[[IsClosed]]가 true이면, 이 단계를 중단한다.
2. newState를 RTCIceGatheringState 열거형에서 설명한 대로 새로운 상태 값을 도출하는 값으로 지정한다.
3. connection의 ICE 수집 상태가 newState와 같으면, 이 단계를 중단한다.
4. connection의 ICE 수집 상태를 newState로 설정한다.
5. connection에서 icegatheringstatechange라는 이벤트를 발생(Fire an event)시킨다.
6. newState가 "complete"이면, connection에 후보 속성이 null로 설정된 RTCPeer Connection IceEvent 인터페이스를 사용하여 icecandidate라는 이벤트를 발생시킨다.

참고

null 후보 이벤트는 레거시 호환성을 보장하기 위해 발생한다. 새 코드는 RTCIceTransport 및(또는) RTCPeerConnection의 수집 상태를 모니터링해야 한다.

ⓓ RTCSessionDescription 설정(Set the RTCSessionDescription)

RTCPeerConnection 객체 connection에 대한 로컬 RTCSessionDescription description을 설정하려면, remote를 false로 설정하여 RTCSessionDescription 알고리즘을 실행한다.

RTCPeerConnection 객체 connection에 대한 원격 RTCSessionDescription description을 설정하려면, remote를 true로 설정하여 RTCSessionDescription 알고리즘을 실행한다.

(remote 부울이 주어진)RTCPeerConnection 객체 connection에 RTCSessionDescription description을 설정하려면 다음 단계를 실행한다:

1. p를 새로운 promise로 지정한다.
2. description.type이 "rollback"이고 connection의 시그널링 상태가 "stable", "have-local-pranswer" 또는 "have-remote-pranswer"이면, 새로 생성된 InvalidState Error로 p를 거절하고 이를 중단한다.
3. jsepSetOfTransceivers를 connection의 트랜시버(transceivers) 세트의 피상적인 복사본(shallow cop)으로 지정한다.
4. 동시에, [JSEP](섹션 5.5. 및 섹션 5.6.)에 설명된 대로 description을 적용하는 프로세스를 시작하고 다음과 같은 추가 제한 사항을 적용한다.
   1) "RtpTransceivers"가 존재하는 것과 관련해서 참의 소스로 jsepSetOfTransceivers를 사용하고, "mid property"로 [[JsepMid]] 내부 슬롯을 사용한다.
   2) 만약 remote가 true이면, 후속 오퍼에 대한 검사를 안정적인 상태로 실행하여 그 사이에 답이 적용된 것처럼 백투백 오퍼(back-to-back offer)의 유효성을 확인한다.
   3) 만약 description을 적용하면 트랜시버 transceiver가 수정되고, transceiver.[[Sender]].[[SendEncodings]]가 비어있지 않고, description 처리로 인한 인코딩과 동일하지 않은 경우, description 적용 프로세스가 실패한다. 이 규격은 원격으로 시작된 RID 재협상을 허용하지 않는다.
   4) 만약 어떤 이유로 description을 적용하는 프로세스가 실패하면, 사용자 에이전트는 아래의 단계를 실행하는 작업을 대기열(queue)에 반드시(MUST) 넣어야 한다.
      (1) 만약, connection.[[IsClosed]]가 true라면, 이 단계를 중단한다.
      (2) [JSEP](섹션 5.5 및 섹션 5.6)에 설명된 연결의 현재 신호 상태에 대해 description.type이 유효하지 않은 경우, 새로 생성된 InvalidStateError로 p를 거부하고 이 단계를 중단한다.

(3) description의 내용이 유효한 SDP 문법이 아닌 경우, RTCError(errorDetail가 "sdp-syntax-error"로 설정되고 sdpLineNumber 속성이 SDP의 라인 넘버로 설정된 상태에서)로 p를 거부하고 이러한 단계를 중단한다.

(4) remote 가 true 이면, connection의 RTCRtcpMuxPolicy가 require이고, 설명(description)이 RTCP mux를 사용하지 않는 경우, 새로 생성된 InvalidAccess Error로 p를 거부하고 이 단계를 중단한다.

(5) 만약 위에 기술한대로, 설명이 RID 재협상을 시도한 경우, 새로 생성된 Invalid AccessError로 p를 거부하고 이 단계를 중단한다.

(6) description이 유효하지 않은 경우, 새로 생성된 InvalidAccessError로 p를 거부하고 이 단계를 중단한다.

(7) 다른 모든 오류의 경우, 새로 생성된 OperationError로 p를 거부한다.

5) description이 성공적으로 적용되면, 유저 에이전트는 다음 단계를 실행하는 작업을 반드시(MUST) 대기열(queue)에 넣는다.

(1) connection.[[IsClosed]]가 true이면, 이 단계를 중단한다.

(2) 만약 remote가 true이고 description이 “offer” 유형인 경우, description을 적용하는 프로세스 중에 성공한 addTrack() 메소드가 있으면 이러한 단계를 중단하고 이전의 성공과 같이 프로세스를 다시 시작하여 프로세스에 추가 트랜시버(transceiver)를 포함한다.

(3) 만약 description이 "offer" 타입이고 connection의 신호 상태가 "stable"이면 connection의 트랜시버(transceiver) 집합에 있는 각 트랜시버에 대해 아래의 단계를 행한다:

1〉 transceiver.[[Sender]].[[LastStableStateSenderTransport]]를 transceiver.[[Sender]].[[SenderTransport]]로 설정한다.

2〉 transceiver.[[Receiver]].[[LastStableStateReceiverTransport]]를 transceiver.[[Receiver]].[[ReceiverTransport]]로 설정한다.

3〉 transceiver.[[Receiver]].[[LastStableStateAssociatedRemoteMedia Streams]]를 transceiver.[[Receiver]].[[AssociatedRemoteMedia Streams]]로 설정한다.

4〉 transceiver.[[Receiver]].[[LastStableStateReceiveCodecs]]를 transceiver.[[Receiver]].[[ReceiveCodecs]]로 설정한다.

(4) 만약 remote가 false이면, 다음 단계 중 하나를 실행한다:

1〉 description이 "offer" 타입인 경우,
connection.[[PendingLocalDescription]]을 description에서 생성된 새 RTCSessionDescription 객체로 설정하고, connection의 신호 상태를 "have-local-offer"로 설정한 다음, 초기 후보를 릴리즈 한다.

2〉 description이 "answer" 타입이면, 제안 답변 협상을 완료한다.
connection.[[CurrentLocalDescription]]을 description에서 생성된 새 RTCSessionDescription 객체로 설정하고,
connection.[[CurrentRemoteDescription]]을
connection.[[PendingRemoteDescription]]로 설정한다.
connection.[[PendingRemoteDescription]] 및
connection.[[PendingLocalDescription]]을 모두 null로 설정한다.
connection.[[LastCreatedOffer]]와 connection.[[LastCreatedAnswer]]을 모두 ""로 설정하고, connection의 신호 상태를 "stable"로 설정하며, 초기 후보를 해제한다. 마지막으로, connection.[[LocalIceCredentialsToReplace]]에 있는 ICE 자격 증명이 description에 없으면, connection.[[LocalIceCredentialsToReplace]]를 빈 세트로 설정한다.

3〉 만약 설명(description)이 “pranswer” 타입인 경우,
connection.[[PendingLocalDescription]]을 description에서 생성된 새 RTCSessionDescription 객체로 설정하고, 연결의 신호 상태를 "have-local-pranswer"로 설정하고, 초기 후보를 릴리즈 한다.

(5) 그렇지 않으면(remote가 true이면, 다음 단계 중 하나를 실행한다:

1〉 description이 “offer” 타입인 경우,
connection.[[PendingRemoteDescription]] 속성을 description에서 생성된 새로운 RTCSessionDescription 객체로 설정하고, connection의 신호 상태를 "have-remote-offer"로 설정한다.

2〉 description이 "answer", 제안(offer) 응답(answer) 협상이 완료된다.
connection.[[CurrentRemoteDescription]]을 description에서 생성된 새 RTCSessionDescription 객체로 설정하고,
connection.[[CurrentLocalDescription]]을
connection.[[PendingLocalDescription]]으로 설정한다.

connection.[[PendingRemoteDescription]]과 connection.[[PendingLocalDescription]]을 모두 null로 설정한다. connection.[[LastCreatedOffer]]와 connection.[[LastCreatedAnswer]]를 모두 ""로 설정하고, connection의 시그널 상태를 "stable"로 설정한다. 마지막으로, connection.[[LocalIceCredentialsToReplace]]의 ICE 자격증명이 새로 설정된 connection.[[CurrentLocalDescription]]에 없는 경우, connection.[[LocalIceCredentialsToReplace]]를 빈 집합으로 설정한다.

3〉 description이 "pranswer" 타입이면, connection.[[PendingRemoteDescription]]을 description에서 생성된 새 RTCSessionDescription 객체로 설정하고 connection의 신호 상태를 "have-remote-pranswer"로 설정한다.

(6) description이 "answer" 타입이고, [SCTP-SDP] 섹션 10.3 및 10.4에 정의된 대로 기존 SCTP 연결의 종결을 시작하는 경우 connection.[[SctpTransport]] 값을 null로 설정한다.

(7) trackEventInits, muteTracks, addList, removeList 및 errorList를 빈 리스트로 설정한다.

(8) 만약, description이 "answer" 또는 "pranswer" 타입이면, 아래의 단계를 실행한다:

1〉 description이 [SCTP-SDP] 섹션 10.3 및 10.4에 정의된 대로 새 SCTP 연결 설정을 시작하면 초기 상태가 “connecting”인 RTCSctpTransport를 만들고 그 결과를 [[SctpTransport]] 슬롯에 할당한다. 그렇지 않으면, SCTP 연결이 설정되었지만, max-message-size SDP 속성이 업데이트 된 경우, connection.[[SctpTransport]]의 데이터 최대 메시지 크기를 업데이트 한다.

2〉 description이 SCTP 전송의 DTLS 역할을 협상하는 경우, 각 RTCDataChannel, channel에 대해 null id를 사용하여, 다음 단계를 실행한다:

① [RTCWEB-DATA-PROTOCOL]에 따라 생성된 새 ID를 channel에 제공한다. 사용 가능한 ID가 생성되지 않으면, channel.[[ReadyState]]를 "closed"로 설정하고, channnel을 errorList에 추가한다.

(9) 만약 description이 "rollback" 타입이 아닌 경우, 다음 단계를 실행한다:

1〉 만약 remote가 false이면, description의 각 미디어 설명에 대해 다음 단계를 실행한다:

① 만약, 미디어 설명이 아직 RTCRtpTransceiver 객체와 연결되지 않은 경우 다음 단계를 실행한다:

ⓐ transceiver를 미디어 설명을 만드는 데 사용되는 RTCRtpTransceiver로 지정한다.

ⓑ transceiver.[[Mid]]를 transceiver.[[JsepMid]]로 설정한다.

ⓒ 만약 transceiver.[[Stopped]]가 true이면, 이 하위 단계들을 중단한다.

ⓓ 만약 미디어 설명이 [BUNDLE]에 따라 기존 미디어 전송을 사용하는 것으로 표시되면, 전송을 해당 전송의 RTP/RTCP 구성 요소를 나타내는 RTCDtlsTransport 객체로 지정한다.

ⓔ 그렇지 않으면, transport를 새로운 기본 RTCIceTransport를 사용하여 새로 생성된 RTCDtlsTransport 객체로 설정한다.

ⓕ transceiver.[[Sender]].[[SenderTransport]]를 transport로 설정한다.

ⓖ transceiver.[[Receiver]].[[ReceiverTransport]]를 transport로 설정한다.

② transceiver를 미디어 설명과 관련된 RTCRtpTransceiver로 지정한다.

③ 만약 transceiver.[[Stopped]]가 true이면, 이 하위 단계들을 중단한다.

④ direction을 미디어 설명의 방향을 나타내는 RTCRtpTransceiverDirection 값으로 지정한다.

⑤ 만약 direction이 "sendrecv" 또는 "recvonly"이면, transceiver.[[Receptive]]를 true로 설정하고, 그렇지 않으면 false로 설정한다.

⑥ transceiver.[[Receiver]].[[ReceiveCodecs]]를 description이 수신을 위한 협상과 현재 유저 에이전트가 수신할 준비가 된 코덱으로 설정한다.

참고

만약 direction이 "sendonly" 또는 "inactive"이면, 수신자는 아무것도 수신할 준비가 되어있지 않으며, 리스트는 비어 있게 된다.

⑦ 만약 description이 "answer" 또는 "pranswer" 타입인 경우, 다음 단계들을 실행한다:

ⓐ description이 전송을 위한 협상과 현재 유저 에이전트가 전송할 수 있는 코덱으로 transceiver.[[Sender]].[[SendCodecs]]를 설정하고, transceiver.[[Sender]].[[LastReturnedParameters]] 를 null로 설정한다.

ⓑ 만약 direction이 "sendonly" 또는 "inactive"이고, transceiver.[[FiredDirection]]은 "sendrecv" 또는 "recvonly"이면, 다음 단계를 실행한다:

- Set the associated remote streams given transceiver.[[Receiver]], an empty list, another empty list, and removeList.transceiver.[[Receiver]], 빈 리스트, 다른 빈 리스트 및 removeList에 연결된 원격 스트림을 설정한다.
- transceiver 및 muteTracks에서 주어진 미디어 설명에 대한 원격 트랙 제거를 처리한다.

ⓒ transceiver.[[CurrentDirection]]와 transceiver.[[FiredDirection]]을 direction으로 설정한다.

2〉 그렇지 않으면(remote가 true인 경우), description의 각 미디어 설명에 대해 다음 단계를 실행한다:

① 만약 description이 "offer" 타입이고 동시 캐스트(simulcast) 수신 요청을 포함하는 경우, 동시 캐스트(simulcast) 속성에 지정된 rid 값 순서를 사용하여 각 동시 캐스트(simulcast) 레이어에 대한 RTCRtpEncoding Parameters 사전을 만들고, 해당 rid 값에 따라 rid 멤버를 채우며, sendEncodings가 생성된 사전을 포함하는 리스트가 되도록 한다.

② supportedEncodings를 구현에서 지원하는 최대의 인코딩 수로 설정한다. sendEncodings의 길이가 supportedEncodings보다 크면, 해당 길이가 supportedEncodings가 되도록 sendEncodings를 줄인다.

③ 만약 sendEncodings이 비어 있지 않으면, 각 인코딩의 scaleResolution DownBy를 2^(sendEncodings 길이 - encoding index - 1)로 설정한다.

④ [JSEP](섹션 5.10.)에 설명되어 있는 대로, 미디어 설명을 나타내는 기존 RTCRtpTransceiver 객체인 transceiver를 찾는다.

⑤ 만약 적절한 트랜시버(transceiver)를 찾았고(트랜시버가 설정됨) sendEncodings가 비어 있지 않으면, transceiver.[[Sender]].[[SendEncodings]]를 sendEncodings로 설정하고, transceiver.[[Sender]].[[LastReturnedParameters]]를 null로 설정한다.

⑥ 만약 적합한 트랜시버를 찾지 못한 경우(transceiver가 설정되지 않음), 다음 단계들을 실행한다:

ⓐ sendEncodings을 사용하여 미디어 설명에서 RTCRtpSender, sender를 만든다.

ⓑ 미디어 설명에서 RTCRtpReceiver, receiver를 만든다.

ⓒ sender, receiver 및 RTCRtpTransceiverDirection 값이 "recvonly"인 RTCRtpTransceiver를 만들고, transceiver를 결과로 둔다.

ⓓ connection의 송수신기(transceivers) 세트에 transceiver를 추가한다.

⑦ 만약 description이 "answer" or "pranswer" 타입이고, transceiver.[[Sender]].[[SendEncodings]].length가 1보다 큰 경우, 다음 단계를 실행한다:

ⓐ 만약 description이 동시 방송(simulcast)이 지원되지 않거나 바람직하지 않음을 나타내면, transceiver.[[Sender]].[[SendEncodings]]에서 첫 번째 사전을 제외한 모든 사전을 제거하고 하위 단계들을 중단한다.

ⓑ 만약 description이 제공된 레이어 중 하나를 거부하면, transceiver.[[Sender]].[[SendEncodings]]에서 거부된 레이어에 해당하는 사전을 제거한다.

ⓒ transceiver.[[Sender]].[[SendEncodings]]의 해당 사전의 활성 멤버를 일시 중지되지 않은 경우 true로 설정하고 일시 중지된 경우 false로 설정하여 각 동시 방송(simulcast) 계층의 [MMUSIC-SIMULCAST]에 표시된 일시 중지된 상태를 업데이트 한다.

⑧ transceiver.[[Mid]]를 transceiver.[[JsepMid]]로 설정한다.

⑨ direction을 미디어 설명의 방향을 나타내는 RTCRtpTransceiverDirection 값으로 지정하지만, 이 피어의 관점을 나타내기 위해 전송 및 수신 방향을 반대로 한다. 미디어 설명이 거부되면 direction을 비활성("inactive")으로 설정한다.

⑩ 만약direction이 "sendrecv" 또는 "recvonly"인 경우, msids를 미디어 설명에서 transceiver.[[Receiver]].[[ReceiverTrack]]과 연관되어 있음을 나타내는 MSID 리스트가 되도록 한다. 그렇지 않으면 msids를 빈 리스트로 둔다.

참고

미디어 설명이 거부된 경우 여기에서 msids는 빈 리스트가 된다.

⑪ transceiver, direction, msids, addList, removeList 및 trackEventInits를 사용하여 원격 트랙을 처리한다.

⑫ transceiver.[[Receiver]].[[ReceiveCodecs]]를 description 수신을 위해 협상하고 유저 에이전트가 현재 수신할 준비가 된 코덱으로 설정한다.

⑬ 만약 description이 "answer" 또는 "pranswer" 타입이면, 다음 단계를 실행한다:

ⓐ transceiver.[[Sender]].[[SendCodecs]]를 description이 전송을 위해 협상하고 현재 유저 에이전트가 전송할 수 있는 코덱으로 설정한다.

ⓑ transceiver.[[CurrentDirection]] 및 transceiver.[[Direction]]를 direction으로 설정한다.

ⓒ [BUNDLE]에 따라, transceiver를 transport의 관련 미디어 설명에서 사용하는 미디어 전송의 RTP/RTCP 구성 요소를 나타내는

RTCDtlsTransport 객체가 transceiver가 되도록 한다.

ⓓ transceiver.[[Sender]].[[SenderTransport]]를 transport로 설정한다.

ⓔ transceiver.[[Receiver]].[[ReceiverTransport]]를 transport로 설정한다.

ⓕ [RFC8445]의 규칙에 따라 transport의 [[IceRole]]을 설정한다.

참고

여기에 적용되는 [RFC8445]의 규칙은 다음과 같다:

- 만약 [[IceRole]]이 unknown이 아니면, [[IceRole]]을 수정하지 않는다.
- 만약 description이 로컬 offer이면, controlling으로 설정한다.
- 만약 description이 원격 오퍼이고, a=ice-lite를 포함하는 경우, [[IceRole]]를 controlling으로 설정한다.
- 만약 description이 원격 오퍼이고 a=ice-lite를 포함하지 않는 경우, [[IceRole]]을 controlled로 설정한다.

이렇게 하면 첫 번째 오퍼가 처리된 후 [[IceRole]]이 항상 값을 갖게된다.

⑭ 만약 미디어 설명이 거부되고, transceiver.[[Stopped]]가 false이면, RTCRtpTransceiver transceiver를 중지한다.

(10) 그렇지 않으면, (description이 "rollback" 타입인 경우) 다음 단계를 실행한다:

1〉 For each transceiver in the connection's set of transceivers run the following steps:
connection's의 트랜시버 세트에 있는 각 transceiver에 대해 다음 단계를 실행한다:

① 만약 transceiver가 롤백 중인 RTCSessionDescription에 적용하기 위한 미디어 설명과 연관되어 있지 않은 경우 연결을 해제 하고 transceiver.[[JsepMid]] 및 transceiver.[[Mid]] 모두 null로 설정한다.

② transceiver.[[Sender]].[[SenderTransport]]를 transceiver.[[Sender]].[[LastStableStateSenderTransport]]로 설정한다.

③ transceiver.[[Receiver]].[[ReceiverTransport]]를 transceiver.[[Receiver]].[[LastStableStateReceiverTransport]]로 설정한다.

④ transceiver.[[Receiver]].[[ReceiveCodecs]]를 transceiver.[[Receiver]].[[LastStableStateReceiveCodecs]]로 설정한다.

⑤ 만약 connection의 신호 상태가 "have-remote-offer"인 경우, 다음 아래 단계를 실행한다:

ⓐ msids를 transceiver.[[Receiver]].[[LastStableStateAssociatedRemoteMediaStreams]]에 있는 모든 MediaStream 객체의 id 리스트이거나, 없는 경우 빈 리스트로 지정한다.

ⓑ transceiver, transceiver.[[CurrentDirection]], msids, addList, removeList, 및 trackEventInits을 사용하여 원격 트랙을 처리한다.

⑥ 만약 롤백 중인 RTCSessionDescription을 적용하여 transceiver가 생성되었고, addTrack()을 통해 트랙이 첨부된 적이 없는 경우, RTCRtpTransceiver transceiver를 중지하고, connection의 트랜시버 세트에서 제거한다.

2〉 connection.[[PendingLocalDescription]]과 connection.[[PendingRemoteDescription]]을 null로 설정하고, connection의 신호 상태를 "stable"로 설정한다.

(11) 만약 description이 "answer" 타입이면, 다음 단계를 실행한다:

1〉 connection의 트랜시버 세트에 있는 각 transceiver에 대해 다음 단계를 실행한다:

① 만약 transceiver가 stopped이고, m= 섹션과 연결되어 있고 연결된 m= 섹션이 connection.[[CurrentLocalDescription]] 또는 connection.[[CurrentRemoteDescription]]에서 거부된 경우, connection의 트랜시버의 세트에서 transceiver를 제거한다.

(12) connection의 신호 상태가 이제 "stable"이면, 다음 단계를 실행한다:

1〉 이전 단계에서 트랜시버 세트에서 제거된 모든 transceiver의 경우, 해당 전송(transceiver.[[Sender]].[[SenderTransport]] 또는 transceiver.[[Receiver]].[[ReceiverTransport]])이 아직 닫히지 않고 논스톱 트랜시버에서 더 이상 참조되지 않는 경우, RTCDtlsTransport 및 관련

RTCIceTransport를 닫는다. 이로 인해 대기 중인 작업의 이러한 객체에서 이벤트가 발생한다.

2〉 협상-필요 플래그를 지우고 협상-필요 플래그를 업데이트 한다.

(13) 만약 위와 같이 connection의 시그널링 상태가 변경되면, connection에서 signalingstatechange이라는 이벤트를 발생시킨다.

(14) errorList의 각 channel에 대해, channel에서 errorDetail 속성이 "data-channel-failure"로 설정된 RTCErrorEvent 인터페이스를 사용하여 error라는 이벤트를 발생시킨다.

(15) muteTracks의 각 track에 대해, track의 음소거 상태를 true 값으로 설정한다.

(16) removeList의 각 stream 및 track 쌍에 대해, stream에서 track 트랙을 제거한다.

(17) addList의 각 stream 및 track 쌍에 대해, stream에 track 트랙을 추가한다.

(18) trackEventInits의 각 항목 entry에 대해, receiver 속성이 entry.receiver로 초기화된 RTCTrackEvent 인터페이스를 사용하여 track 이라는 이벤트를 발생시키며, 해당 streams 속성이 entry.streams로 초기화되고 transceiver 속성 connection 객체에서 entry.transceiver로 초기화 되었다.

(19) undefined로 p를 해결한다.

5. p를 리턴한다.

ⓔ 구성 설정 (Set the configuration)

구성을 설정(set a configuration)하기 위해 다음 단계를 실행한다:

1. 처리를 위해 configuration을 RTCConfiguration 사전으로 지정한다.
2. connection을 대상(target) RTCPeerConnection 객체로 지정한다.
3. configuration.certificates가 설정된 경우, 다음 단계를 실행한다:
   1) 만약 configuration.certificates의 길이가 connection.[[Configuration]].certificates의 길이와 다른 경우, connection.[[Configuration]].InvalidModificationError.를 발생시킨다.
   2) index를 0으로 초기화한다.
   3) size를 configuration.certificates의 길이로 초기화한다.
   4) index가 size보다 작으면, 다음 단계를 실행한다:

(1) 만약 index에서 configuration.certificates 값으로 표시되는 ECMAScript 객체가 index에서 connection.[[Configuration]].certificates 값으로 표시되는 ECMAScript 객체와 같지 않으면, InvalidModificationError를 발생(throw)시킨다.

(2) index를 1 증가 시킨다.

4. 만약 configuration.bundlePolicy의 값이 설정되어 있고 해당 값이 connection의 번들 정책(bundle policy)과 다르면, InvalidModificationError 를 발생시킨다.
5. 만약 configuration.rtcpMuxPolicy의 값이 설정되어 있고 해당 값이 connection의 rtcpMux 정책(rtcpMux policy)과 다르면, InvalidModificationError를 발생시킨다.
6. 만약 configuration.iceCandidatePoolSize의 값이 설정되어 있고 해당 값이 connection의 이전에 설정된 iceCandidatePoolSize와 다르고, setLocalDescription이 이미 호출된 경우, InvalidModificationError를 발생시킨다.
7. ICE 에이전트의 ICE 전송 설정을 configuration.iceTransportPolicy의 값으로 설정한다. [JSEP] (섹션 4.1.16)의 정의와 같이, 새 ICE 전송 설정이 기존 설정을 변경하면, 다음 수집 단계까지 아무런 조치도 취하지 않는다. 스크립트가 즉시 수행하려면, ICE를 다시 시작해야 한다.
8. [JSEP] (섹션 3.5.4 및 섹션 4.1.1)의 정의와 같이, ICE 에이전트의 프리 패치된(prefetched) ICE 후보 풀 크기를 configuration.iceCandidatePoolSize의 값으로 설정한다. 새 ICE 후보 풀 크기가 기존 설정을 변경하는 경우, [JSEP](섹션 4.1.16)의 정의와 같이, 새 풀링된 후보를 즉시 수집하거나, 기존 풀링된 후보를 폐기할 수 있다.
9. validatedServers를 빈 리스트로 둔다.
10. 만약 configuration.iceServers가 정의 된 경우, 각 요소에 대해 다음 단계를 실행한다:
    1) server를 현재 리스트 요소로 설정한다.
    2) urls을 server.urls로 지정한다.
    3) 만약 urls가 문자열이면, 해당 문자열로만 구성된 리스트로 urls을 설정한다.
    4) 만약 urls이 비어 있으면, SyntaxError를 발생시킨다.
    5) urls의 각 url에 대해 다음 단계를 실행한다:
        (1) [RFC3986]에 정의된 일반적인 URI 문법을 사용해서 url을 분석하고 scheme name을 가져온다. [RFC3986]에 정의된 문법에 기반한 분석이 실패하면, SyntaxError를 발생시킨다. scheme name이 turn이거나 turns이고, [RFC7065]

의 정의와 같은 문법을 사용하여 url 분석이 실패하면, SyntaxError를 발생시킨다. scheme name 이 stun이거나 stuns이고, [RFC7064]의 정의와 같은 문법을 사용하여 url의 분석이 실패하면 SyntaxError를 발생시킨다.

(2) 만약 scheme name이 turn이거나 turns이고, server.username나 server.credential 중 하나가 생략된 경우, InvalidAccessError를 발생시킨다.

(3) 만약 scheme name이 turn이거나 turns이고, server.credentialType이 "password"이며, server.credential이 DOMString이 아닌 경우, nvalidAccessError를 발생시킨다.

6) validatedServers에 server를 추가한다.

11. ICE 에이전트의 ICE 서버 리스트를 validatedServers로 설정한다.
[JSEP](섹션 4.1.16)의 정의와 같이, 새로운 서버 리스트가 ICE 에이전트의 기존 ICE 서버 리스트를 대체하는 경우, 다음 수집 단계까지 아무런 조치도 취하지 않는다. 스크립트를 즉시 수행하려면, ICE 재시작해야 한다. 그러나, ICE 후보 풀의 크기가 0이 아닌 경우, 기존 풀링된 후보는 모두 폐기되고, 새 후보는 새로운 서버에서 수집된다.

12. [[Configuration]] 내부 슬롯에 configuration을 저장한다.

② 인터페이스 정의

이 섹션에 제시된 RTCPeerConnection 인터페이스는 이 규격 전반에 걸쳐 여러 부분 인터페이스에 의해 확장된다. 특히, MediaStreamTrack 객체를 송수신하기 위한 API를 추가하는 RTP 미디어 API 섹션이 있다.

WebIDL

```
[Exposed=Window]
interface RTCPeerConnection : EventTarget {
constructor(optional RTCConfiguration configuration = {});
Promise<RTCSessionDescriptionInit> createOffer(optional RTCOfferOptions
options = {});
Promise<RTCSessionDescriptionInit> createAnswer(optional RTCAnswerOptions
options = {});
```

```
Promise<undefined> setLocalDescription(optional
RTCLocalSessionDescriptionInit description = {});
    readonly attribute RTCSessionDescription? localDescription;
    readonly attribute RTCSessionDescription? currentLocalDescription;
    readonly attribute RTCSessionDescription? pendingLocalDescription;
Promise<undefined> setRemoteDescription(RTCSessionDescriptionInit description);
    readonly attribute RTCSessionDescription? remoteDescription;
    readonly attribute RTCSessionDescription? currentRemoteDescription;
    readonly attribute RTCSessionDescription? pendingRemoteDescription;
    Promise<undefined> addIceCandidate(optional RTCIceCandidateInit candidate = {});
    readonly attribute RTCSignalingState signalingState;
    readonly attribute RTCIceGatheringState iceGatheringState;
    readonly attribute RTCIceConnectionState iceConnectionState;
    readonly attribute RTCPeerConnectionState connectionState;
    readonly attribute boolean? canTrickleIceCandidates;
    undefined restartIce();
RTCConfiguration getConfiguration();
undefined setConfiguration(optional RTCConfiguration configuration = {});
undefined close();
    attribute EventHandler onnegotiationneeded;
    attribute EventHandler onicecandidate;
    attribute EventHandler onicecandidateerror;
    attribute EventHandler onsignalingstatechange;
    attribute EventHandler oniceconnectionstatechange;
    attribute EventHandler onicegatheringstatechange;
    attribute EventHandler onconnectionstatechange;

// 레거시 인터페이스 확장
// 이 섹션의 메소드 지원은 선택사항이다.
// 만약 이런 메소드가 지원되는 경우
```

```
// "레거시 인터페이스 확장" 섹션에서
// 정의된 대로 구현되어야 한다.

Promise<undefined> createOffer(RTCSessionDescriptionCallback successCallback,
                               RTCPeerConnectionErrorCallback failureCallback,
                               optional RTCOfferOptions options = {});
Promise<undefined> setLocalDescription(optional
                               RTCLocalSessionDescriptionInit description = {},
                               VoidFunction successCallback,
                               RTCPeerConnectionErrorCallback failureCallback);
Promise<undefined> createAnswer(RTCSessionDescriptionCallback
                               successCallback,
                               RTCPeerConnectionErrorCallback failureCallback);
Promise<undefined> setRemoteDescription(RTCSessionDescriptionInit
                               description,
                               VoidFunction successCallback,
                               RTCPeerConnectionErrorCallback failureCallback);
Promise<undefined> addIceCandidate(RTCIceCandidateInit candidate,
                               VoidFunction successCallback,
                               RTCPeerConnectionErrorCallback failureCallback);
};
```

속성(Attributes)

localDescription, RTCSessionDescription 타입, 읽기전용, null 허용

localDescription 속성은 null이 아니면 [[PendingLocalDescription]]을 반드시 (MUST) 리턴하고 그렇지 않으면 [[CurrentLocalDescription]]을 반드시(MUST) 리턴해야 한다.

참고로 [[CurrentLocalDescription]].sdp 와 [[PendingLocalDescription]].sdp는 해당 setLocalDescription 호출에 전달된 SDP 값과 문자열 방식으로 동일할 필요가 없다 (예: SDP가 구문 분석(parse) 및 재포맷 될 수 있으며, ICE 후보가 추가될 수 있음).

currentLocalDescription, RTCSessionDescription 타입, 읽기전용, null 허용

currentLocalDescription 속성은 [[CurrentLocalDescription]]을 반드시(MUST) 리턴해야 한다.

RTCPeerConnection이 마지막으로 안정 상태로 전환되었을 때 성공적으로 협상된 로컬 설명과 제안 혹은 답변이 생성된 이후 ICE 에이전트에 의해 생성된 로컬 후보를 나타낸다.

pendingLocalDescription, RTCSessionDescription 타입, 읽기전용, null 허용

pendingLocalDescription 속성은 [[PendingLocalDescription]]을 반드시(MUST) 반환해야 한다.

협상 과정에 있는 지역 설명과 제안(offer) 또는 답변(answer)이 생성된 이후 ICE 에이전트가 생성한 지역 후보를 나타낸다. RTCPeerConnection이 안정된 상태이면, 값은 null이다.

remoteDescription, RTCSessionDescription 타입, 읽기전용, null 허용

remoteDescription 속성은 null이 아니면 [[PendingRemoteDescription]]을 반드시(MUST)반환해야 하고 그렇지 않으면 [[CurrentRemoteDescription]]를 반드시(MUST) 반환해야 한다.

참고로 [[CurrentRemoteDescription]].sdp와 [[PendingRemoteDescription]]. sdp는 해당 setRemoteDescription 호출에 전달된 SDP 값과 문자열-방식으로 동일할 필요가 없다(예. SDP를 구문 분석과 형식 변경을 할 수 있으며, ICE 후보가 추가될 수 있음).

currentRemoteDescription, RTCSessionDescription 타입, 읽기전용, null 허용

currentRemoteDescription 속성은 [[CurrentRemoteDescription]]을 반드시(MUST) 반환해야 한다.

이 설명은 RTCPeerConnection이 안정 상태로 전환되었을 때 제안(offer) 또는 답변 (answer)이 생성된 이후 addIceCandidate()를 통해 제공된 모든 원격 후보와 성공적으로 협상된 마지막 원격 설명을 나타낸다.

pendingRemoteDescription, RTCSessionDescription 타입, 읽기전용, null 허용

pendingRemoteDescription 속성은 [[PendingRemoteDescription]]을 반드시(MUST) 반환해야 한다.

협상중인 원격 설명을 나타내며, 제안(offer) 또는 답변(answer)이 생성된 이후 addIce Candidate()를 통해 제공된 원격 후보로 완료된다. RTCPeerConnection이 안정적인 상태일

경우, 값은 null이다.

signalingState, RTCSignalingState 타입, 읽기전용

signalingState 속성은 RTCPeerConnection 객체의 신호 상태를 반드시(MUST) 리턴해야 한다.

iceGatheringState, RTCIceGatheringState 타입, 읽기전용

iceGatheringState 속성은 RTCPeerConnection 인스턴스의 ICE 수집 상태를 반드시 (MUST) 리턴해야 한다.

iceConnectionState, RTCIceConnectionState 타입, 읽기전용

iceConnectionState 속성은 RTCPeerConnection 인스턴스의 ICE 연결 상태를 반드시 (MUST) 리턴해야 한다.

connectionState, RTCPeerConnectionState 타입, 읽기전용

connectionState 속성은 RTCPeerConnection 인스턴스의 연결 상태를 반드시(MUST) 리턴해야 한다.

canTrickleIceCandidates, boolean 타입, 읽기전용, null 허용

canTrickleIceCandidates 속성은 원격 피어가 trickled ICE 후보 [TRICKLE-ICE]를 수락할 수 있는지 여부를 나타낸다. 이 값은 [JSEP] (섹션 4.1.15.)에 정의와 같이 원격 설명이 trickle ICE에 대한 지원을 나타내는지 여부에 따라 결정된다.

setRemoteDescription 완료하기 전에 이 값은 null이다.

onnegotiationneeded, EventHandler 타입

이 이벤트 핸들러의 이벤트 타입은 negotiationneeded이다.

onicecandidate, EventHandler 타입

이 이벤트 핸들러의 이벤트 타입은 icecandidate이다.

onicecandidateerror, EventHandler 타입

이 이벤트 핸들러의 이벤트 타입은 icecandidateerror이다.

onsignalingstatechange, EventHandler 타입

이 이벤트 핸들러의 이벤트 타입은 signalingstatechange이다.

oniceconnectionstatechange, EventHandler 타입

이 이벤트 핸들러의 이벤트 타입은 iceconnectionstatechange이다.

onicegatheringstatechange, EventHandler 타입

이 이벤트 핸들러의 이벤트 타입은 icegatheringstatechange이다.

onconnectionstatechange, EventHandler 타입

이 이벤트 핸들러의 이벤트 타입은 connectionstatechange이다.

메소드(Methods)

createOffer

이 createOffer 메소드는 RTCPeerConnection에 첨부된 로컬 MediaStreamTrack에 대한 설명을 포함하여 세션에 대해 지원되는 구성과 함께 RFC 3264 제안을 포함하는 SDP의 blob을 생성한다. 여기에서 지원하는 codec/RTP/RTCP 기능, ICE 에이전트 및 DTLS 연결의 구현 및 인자, 생성된 오퍼에 대한 추가 제어를 제공하기 위해 options 인자를 제공할 수 있다.

시스템에 제한된 리소스(예. 한정된 수의 디코더)가 있는 경우, createOffer는 시스템의 현재 상태를 반영하는 제안을 리턴해야 한다. 그러면 setLocalDescription이 해당 리소스를 획득하려고 할 때 성공할 것이다. 세션 설명은 최소한 반환된 promise의 이행 콜백이 끝날 때까지 오류를 일으키지 않고 setLocalDescription으로 반드시(MUST) 사용할 수 있어야 한다.

SDP 생성은 [JSEP]에 설명된 제안을 생성하기 위한 적절한 프로세스를 따라야 한다. 단, 유저 에이전트는 이 경우 JSEP의 목적을 위해 stopping 트랜시버를 stopped로 반드시 (MUST) 취급해야 한다.

제안으로서, 생성된 SDP에는 세션에서 지원하거나 선호하는 codec/RTP/RTCP 기능의 전체 세트가 포함된다(답변과 반대로, 사용할 특정 협상 하위 집합만 포함). 세션이 설정된 후 createOffer가 호출되는 경우, createOffer는 현재 세션과 호환되는 오퍼를 생성하여 트랙의 추가 또는 제거와 같이 마지막 완전한 오퍼-답변 교환 이후 세션에 반영된 모든 변경 사항을 통합한다. 변경 사항이 없는 경우, 제안에는 현재 지역 설명의 기능과 업데이트된 제안에서 협상할 수 있는 추가 기능이 포함된다.

생성된 SDP는 ICE 에이전트의 usernameFragment, password 및 ICE 옵션([ICE] 섹션 14에 정의됨)도 포함되며, 에이전트가 수집한 모든 로컬 후보도 포함될 수 있다.

RTCPeerConnection에 대한 configuration의 certificates 값은 RTCPeerConnection에 대해 애플리케이션에서 구성한 인증서를 제공한다. 이러한 인증서는 기본 인증서와 함께 인증서 지문을 생성하는 데 사용된다. 이 인증서 지문은 SDP 구성에 사용된다.

SDP를 생성하는 프로세스는 기본 시스템의 미디어 기능의 하위 집합을 노출하며, 일반적으로 장치에 지속적인 상호 출처 정보를 제공한다. 따라서 애플리케이션의 지문 표면이 증가한다. 프라이버시에 민감한 상황에서 브라우저는 기능의 공통 부분 집합만 일치하는 SDP 생성과 같은 완화를 고려할 수 있다.

메소드가 호출되면, 유저 에이전트는 다음 단계를 반드시(MUST) 실행해야 한다:

1. connection을 메소드가 호출된 RTCPeerConnection 객체로 지정한다.
2. connection.[[IsClosed]]가 true이면, 새로 생성된 InvalidStateError와 함께 거부된 promise를 반환한다.
3. 오퍼 생성 결과를 connection의 오퍼레이션 체인에 connection으로 연결한 결과를 리턴한다.

제공된 connection 오퍼 작성을 위해 다음 단계를 실행한다:

1. 만약 connection의 시그널링 상태가 "stable"도 "have-local-offer"도 아닌 경우, 새로 생성된 InvalidStateError와 함께 거부된 promise를 리턴한다.
2. p를 새로운 promise로 지정한다.
3. 동시에, connection 및 p가 주어진 제안(offer)을 생성하기 위해 병렬 단계를 시작한다.
4. p를 리턴한다.

제공된 connection 및 a promise p 제안(offer)을 생성하는 병렬 단계는 다음과 같다:

1. 만약 connection이 인증서 세트로 구성되지 않았고, 아직 생성되지 않은 경우, 생성될 때까지 기다린다.
2. [JSEP] (섹션 4.1.6)에 설명 된대로, 제안자의 시스템 상태를 검사하여 제안 생성에 필요한 현재 사용 가능한 리소스를 확인한다.
3. 만약 어떤 이유로든 이 검사가 실패하면, 새로 생성된 OperationError로 p를 거부하고 이 단계를 중단한다.

4. connection and p가 제공된 오퍼를 작성하기 위해 마지막 단계를 실행하는 태스크를 큐에 넣는다.

connection 및 promise p가 제공된 오퍼를 생성하는 마지막 단계는 다음과 같다:

1. 만약 connection.[[IsClosed]]가 true이면, 이 단계를 중단한다.
2. 만약 제안자의 시스템 상태에 대한 추가 검사가 필요한 방식으로 connection이 수정된 경우, 병행적인 병렬 단계를 시작하여 connection 및 p가 주어지면 제안을 다시 만들고, 이 단계를 중단한다.

참고

예를 들어, 오디오 RTCRtpTransceiver만 connection에 추가 되었을 때 create Offer가 호출되었지만, 제안을 생성하기 위해 병렬 단계를 수행하는 동안, 비디오 RTCRtp Transceiver가 추가되어, 비디오 시스템 리소스의 추가 검사가 필요한 경우 필요할 수 있다.

3. 이전 검사에서 얻는 정보를 바탕으로, 연결의 현재 상태와 RTCRtpTransceiver가 [JSEP] (섹션 5.2)에 설명된 대로 SDP 제안, sdpString을 생성한다.
   1) [BUNDLE] (섹션 7)에 설명한 바와 같이, 번들링을 사용하는 경우(RTCBundlePolicy 참조) BUNDLE 그룹을 협상하기 위해 반드시 m= 섹션 태그가 있는 제공자를 선택해야 한다. 유저 에이전트는 m= 섹션으로 태그가 지정된 트랜시버 세트의 첫 번째 논스톱 트랜시버에 해당하는 m= 섹션을 반드시 선택해야 한다. 이를 통해 원격 엔드 포인트는 SDP를 구문 분석하지 않고도 어떤 트랜시버가 제공자 태그가 붙은 m= 섹션인지 예측할 수 있다.
   2) 미디어 설명과 관련된 트랜시버의 codec preferences는 다음과 같은 필터링이 적용된 RTCRtpTransceiver.[[PreferredCodecs]]의 값이라고 한다(또는 [[PreferredCodecs]]가 비어있는 경우 설정되지 않음).
      (1) 만약 direction이 "sendrecv"인 경우, RTCRtpSender.getCapabilities(kind).codecs 및 RTCRtpReceiver.getCapabilities(kind).codecs의 교차점에 포함되지 않은 코덱을 제외한다.
      (2) 만약 direction이 "sendonly"인 경우, RTCRtpSender.getCapabilities(kind).codecs에 포함되지 않은 코덱을 제외한다.

(3) direction이 "recvonly"인 경우, RTCRtpReceiver.getCapabilities(kind).codecs에 포함되지 않은 코덱을 제외한다.

The filtering MUST NOT change the order of the codec preferences.
필터링은 코덱 기본 설정의 순서를 절대로 변경해서는 안 된다.

3) RTCRtpSender의 [[SendEncodings]] 슬롯 길이가 1보다 크면, RTCRtpSender의 [[SendEncodings]]에 지정된 각 인코딩에 대해, 해당 미디어 섹션에 a=rid send 라인을 추가하고 a=simulcast:send 라인은 encodings 필드에 제공된 것과 동일한 순서로 RID를 제공한다. RID 제한은 설정되지 않는다.

참고

[SDP-SIMULCAST] 섹션 5.2는 a=simulcast 라인에서 RID의 순서가 제안된 선호 순서를 제안하도록 지정한다. 브라우저가 모든 인코딩을 전송하지 않기로 결정하면, 먼저 목록의 마지막 인코딩 전송을 중지할 것으로 예상해야 한다.

4) offer를 type 멤버가 문자열 “offer”로 초기화 되고 sdp 멤버가 sdpString으로 초기화된 새로 생성된 RTCSessionDescriptionInit 사전이 되도록 한다.

5) [[LastCreatedOffer]] 내부 슬롯을 sdpString으로 설정한다.

6) offer로 p를 해결한다.

createAnswer

createAnswer 메소드는 원격 구성의 매개 변수와 호환되는 세션에 대해 지원되는 구성으로 [SDP] 응답을 생성한다. createOffer와 마찬가지로, SDP의 반환된 blob에는 이 RTCPeer Connection에 연결된 로컬 MediaStreamTrack, 이 세션에 대해 협상된 codec/RTP/RTCP 옵션 및 ICE 에이전트가 수집한 모든 후보에 대한 설명이 포함된다. 생성된 답변에 대한 추가 제어를 제공하기 위해 옵션 매개 변수를 제공 할 수 있다.

createOffer와 마찬가지로, 반환된 설명은 시스템의 현재 상태를 반드시 반영해야 한다. 세션 설명은 적어도 반환된 promise의 이행 콜백이 끝날 때까지 오류를 일으키지 않고 setLocal Description에 의해 사용할 수 있어야 한다.

응답(answer)으로, 생성된 SDP는 해당 오퍼와 함께 미디어 평면(plane)을 설정하는 방법을 지정하는 특정 codec/RTP/RTCP 구성을 포함할 것이다. SDP의 생성은 [JSEP]에 설명된 응답을

생성하기 위한 적절한 프로세스를 따라야 한다.

생성된 SDP에는 ICE 에이전트의 usernameFragment, password 및 ICE 옵션([ICE] 섹션 14에 정의됨)도 포함되며, 에이전트가 수집한 모든 로컬 후보도 포함 할 수도 있다.

RTCPeerConnection에 대한 구성의 certificates 값은 RTCPeerConnection에 대한 애플리케이션에 의해 구성된 인증서를 제공한다. 이 인증서는, 기본 인증서와 함께 인증서 지문 세트를 생성하는 데 사용된다. 이 인증서 지문은 SDP의 구성에 사용된다.

응답은 [JSEP] (섹션 4.1.8.1.)에 설명된 대로, type을 "pranswer"로 설정하여 임시로 표시할 수 있다.

메소드가 호출되면 유저 에이전트는 반드시(MUST) 다음 단계를 실행한다:

1. connection을 메소드가 호출된 RTCPeerConnection 객체로 지정한다.
2. 만약 connection.[[IsClosed]]가 true이면, 새로 생성된 InvalidStateError와 함께 거부된 promise를 반환한다.
3. connection의 오퍼레이션 체인에 connection으로 답변을 생성한 결과를 체인화한 결과를 리턴한다.

지정된 connection 응답을 생성하려면 다음 단계를 실행한다:

1. 만약 connection의 신호 상태가 "have-remote-offer"도 "have-local-pranswer"도 아닌 경우, 새로 생성된 InvalidStateError로 거부된 promise를 리턴한다.
2. p를 새로운 promise로 둔다.
3. 동시에, 병렬 단계를 시작하여 주어진 connection과 p에 대한 응답을 생성한다.
4. p를 리턴한다.

주어진 connection 및 promise p에 대한 응답을 만들기 위한 병렬 단계는 다음과 같다:

1. 만약 connection이 인증서 세트로 구성되지 않았고, 아직 생성되지 않은 경우, 생성될 때까지 기다린다.
2. [JSEP] (섹션 4.1.7.)에 설명된 대로, 응답자의 시스템 상태를 검사하여 응답 생성에 필요한 현재 가용 리소스를 확인한다.
3. 어떤 이유로든 이 검사가 실패하면, 새로 생성된 OperationError로 p를 거부하고 이 단계를 중단한다.
4. 지정된 p에 대한 응답을 생성하려면 최종 단계를 실행하는 작업을 큐에 넣는다.

주어진 promise p에 답을 만들기 위한 최종 단계는 다음과 같다:

1. 만약 connection.[[IsClosed]]가 true이면, 이 단계를 중단한다.
2. 만약 connection이 응답자의 시스템 상태에 대한 추가 검사가 필요한 방식으로 수정된 경우, 병렬 단계를 병렬로 시작하여 주어진 connection과 p에 대해 응답을 다시 만들고, 이 단계를 중단한다.

참고

예를 들어, RTCRtpTransceiver의 direction이 "recvonly"일때 createAnswer가 호출되었지만, 응답을 생성하기 위해 병렬 단계를 수행하는 동안, direction이 "sendrecv"로 변경되어, 비디오 인코딩 리소스에 대한 추가 검사가 필요한 경우가 생길 수 있다.

3. [JSEP](섹션 5.3.)에 설명된 대로, 이전 검사에서 얻은 정보와 connection의 현재 상태 및 해당 RTCRtpTransceiver에 대해, SDP 응답인 sdpString을 생성한다.
   1) m= section에 연결된 트랜시버의 codec preferences는 다음 필터링이 적용된 RTCRtpTransceiver.[[PreferredCodecs]]의 값이라고 한다.
   (혹은 [[PreferredCodecs]]가 비어있는 경우 설정되지 않음.):
      (1) direction이 "sendrecv"인 경우, RTCRtpSender.getCapabilities(kind).codecs와 RTCRtpReceiver.getCapabilities(kind).codecs의 교차점에 포함되지 않은 코덱은 제외한다.
      (2) 만약 direction이 "sendonly"인 경우, RTCRtpSender.getCapabilities(kind).codecs에 포하되지 않은 코덱은 제외한다.
      (3) 만약 direction이 "recvonly"인 경우, RTCRtpReceiver.getCapabilities(kind).codecs에 포함되지 않은 코덱은 제외한다.

      The filtering MUST NOT change the order of the codec preferences.
      필터링은 코덱 기본 설정의 순서를 절대 변경해서는 안 된다. (MUST NOT)
   2) 만약 RTCRtpSender의 [[SendEncodings]] 슬롯의 길이가 1보다 크면, RTCRtpSender의 [[SendEncodings]]에 지정된 각 인코딩에 대해, 해당 미디어 섹션에 a=rid send 라인을 추가하고, encodings 필드에 제공된 것과 동일한 순서로 RID를 제공하는 a=simulcast:send 라인을 추가한다. RID 제한은 설정되지 않았다.

4. answer를 type 멤버가 문자열 "answer"로 초기화 되고 이 sdp 멤버가 sdpString으로 초기화된 새로 생성된 RTCSessionDescriptionInit 사전이 되도록 한다.
5. [[LastCreatedAnswer]] 내부 슬롯을 sdpString로 설정한다.
6. answer로 p를 해결한다.

setLocalDescription

setLocalDescription 메소드는 제공된 RTCLocalSessionDescriptionInit를 로컬 설명으로 적용하도록 RTCPeerConnection에 지시한다.

이 API는 로컬 미디어 상태를 변경한다. 애플리케이션이 하나의 미디어 형식에서 다른 호환되지 않는 형식으로 변경하려는 시나리오를 성공적으로 처리하기 위해, RTCPeer Connection은 최종 답변을 수신할 때까지 현재 및 보류중인 로컬 설명(예. 두 설명에 존재하는 지원 코덱)의 사용을 동시에 지원할 수 있어야 한다. 이때 RTCPeerConnection은 보류중인 로컬 설명을 완전히 채택하거나, 원격에서 변경을 거부한 경우 현재 설명으로 롤백할 수 있다.

설명 전달은 선택 사항이다. 생략된 경우, setLocalDescription이 암묵적으로 제안을 생성하거나 필요에 따라 답변을 생성한다. [JSEP] (섹션 5.4.)에서 언급한 바와 같이, SDP에 대한 설명이 전달될 경우, 해당 SDP는 createOffer 또는 createAnswer에서 리턴되었을 때와 다르게 변경될 수 없다.

메소드가 호출되면, 유저 에이전트는 다음 단계를 반드시(MUST) 실행해야 한다.

1. description을 메소드의 첫 인자로 지정한다.
2. connection을 메소드가 호출된 RTCPeerConnection 객체로 지정한다.
3. sdp를 description.sdp로 지정한다.
4. connection의 작업 체인에 다음 단계를 연결한 결과를 리턴한다:
    1) type이 있는 경우 description.type, 없는 경우 "offer"로 하고 connection의 신호 상태는 "stable", "have-local-offer" 또는 "have-remote-pranswer"로 하고, 그렇지 않으면 "answer"로 한다.
    2) 만약 type이 "offer"이고, sdp가 빈 문자열이 아니고 connection.[[LastCreated Offer]]과 같지 않으면, 새로 생성된 InvalidModificationError로 거부된 promise를 리턴하고 이 단계를 중단한다.

3) 만약 type이 "answer" 또는 "pranswer"이고, sdp가 빈 문자열이 아니고 connection.[[LastCreatedAnswer]]와 같지 않으면, 새로 생성된 InvalidModificationError로 거부된 promise를 리턴하고 이 단계를 중단한다.

4) 만약 sdp가 빈 문자열이고, type이 "offer"이면, 다음 아래 단계를 실행한다:

   (1) sdp를 connection.[[LastCreatedOffer]]의 값으로 설정한다.

   (2) 만약 sdp가 빈 문자열이거나, 제공자의 connection 시스템 상태를 더 이상 정확하게 나타내지 않는 경우, p를 connection로 제안을 생성한 결과가 되게 하고, 첫 번째 인자로 표시된 로컬 RTCSessionDescription을 설정하는 이행 단계로 p에 대한 반응 결과를 리턴한다.

5) 만약 sdp가 빈 문자열이고, 타입이 "answer" 또는 "pranswer"인 경우, 다음 아래 단계를 실행한다:

   (1) sdp를 connection.[[LastCreatedAnswer]]의 값으로 설정한다.

   (2) 만약 sdp가 빈 문자열이거나, 응답자의 connection 시스템 상태를 더 이상 정확하게 나타내지 않는 경우, p를 connection으로 답변을 생성한 결과로 두고, 아래와 같은 이행 단계를 통해 p에 반응한 결과를 리턴한다.

      1〉 answer을 이 이행 단계에 대한 첫 번째 인자로 둔다.

      2〉 {type, answer.sdp}로 표시된 로컬 RTCSessionDescription 설정 결과를 리턴한다.

6) {type, sdp}으로 표시된 로컬 RTCSessionDescription 설정 결과를 리턴한다.

참고

[JSEP] (섹션 5.9.)에서 언급한 바와 같이, 이 메소드를 호출하면 ICE 에이전트에 의한 ICE 후보 수집 프로세스가 트리거 될 수 있다.

setRemoteDescription

setRemoteDescription 메소드는 제공된 RTCSessionDescriptionInit를 원격 오퍼 또는 응답으로 적용하도록 RTCPeerConnection에 지시한다. 이 API는 로컬 미디어 상태를 변경한다.

메소드가 호출되면, 유저 에이전트는 다음 단계를 실행한다:

1. description을 메소드의 첫 번째 인자로 지정한다.

2. connection을 메소드가 호출된 RTCPeerConnection 객체로 설정한다.
3. connection의 작업 체인에 다음 단계를 연결한 결과를 리턴한다.
    1) 만약description.type이 "offer"이고 [JSEP] (섹션 5.5. 및 5.6)에 설명된 대로 connection의 현재 신호 상태에 대해 유효하지 않은 경우, 다음 아래 단계를 실행한다:
        (1) p를 로컬 RTCSessionDescription을 {type: "rollback"}으로 표시한 결과로 둔다.
        (2) 원격 RTCSessionDescription description을 설정하는 이행 단계로 p에 응답한 결과를 리턴하고, 이 단계를 중단한다.
    2) 원격 RTCSessionDescription description 설정 결과를 리턴한다.

addIceCandidate

addIceCandidate 메소드는 ICE 에이전트에 원격 후보를 제공한다. 이 메소드는 candidate 멤버에 대해 빈 문자열을 호출될 때 원격 후보의 끝을 나타내는데 사용할 수도 있다. 이 메소드에서 사용하는 인자의 유일한 멤버는 candidate, sdpMid, sdpMLineIndex 및 username Fragment이다. 나머지는 무시된다. 메소드가 호출되면, 유저 에이전트는 다음 단계를 반드시 (MUST) 실행해야 한다.

1. candidate를 메소드의 인자로 둔다.
2. connection을 메소드가 호출된 RTCPeerConnection 객체로 둔다.
3. 만약 candidate.candidate가 빈 문자열이 아니고 candidate.sdpMid와 candidate.sdpMLineIndex가 모두 null이면, 새로 생성된 TypeError와 함께 거부된 promise를 리턴한다.
4. connection의 작업 체인에 다음 단계를 연결한 결과를 리턴한다.
    1) 만약 remoteDescription이 null이면 새로 생성된 InvalidStateError와 함께 거부된 promise를 리턴한다.
    2) 만약 candidate.sdpMid가 null이 아니면, 다음 단계를 실행한다:
        (1) 만약 candidate.sdpMid가 remoteDescription의 미디어 설명 중 중간과 같지 않으면, 새로 생성된 OperationError와 함께 거부된 promise를 리턴한다.
    3) 그렇지 않고, candidate.sdpMLineIndex가 null이 아니면, 다음 단계를 실행한다:
        (1) 만약 candidate.sdpMLineIndex가 remoteDescription의 미디어 설명 수보다 같으면, 새로 생성된 OperationError와 함께 거부된 promise를 리턴한다.

4) 만약 candidate.sdpMid 또는 candidate.sdpMLineIndex중 하나가 remote Description에서 관련 트랜시버가 stopped인 미디어 설명을 나타내는 경우, undefined로 해결된 promise를 리턴한다.

5) 만약 candidate.usernameFragment가 null이 아니고, 적용된 원격 설명의 해당 미디어 설명에 있는 사용자명 조각과 동일하지 않으면, 새로 생성된 OperationError와 함께 거부된 promise를 리턴한다.

6) p를 새로운 promise로 둔다.

7) 동시에, 후보자가 시스템 관리적으로 금지(administratively prohibited)되지 않는다면, [JSEP] (섹션 4.1.17.)에 설명된 대로 ICE 후보 candidate를 추가한다.
candidate.usernameFragment를 사용해서 ICE 생성을 식별한다; username Fragment가 null이면, 최신 ICE 생성에 대한 candidate를 처리한다.
만약 candidate.candidate가 빈 문자열이면, candidate를 해당 미디어 설명 및 ICE 후보 생성을 위한 후보 끝 표시로 처리한다. candidate.sdpMid 및 candidate.sdpMLineIndex가 모두 null이면, 이는 모든 미디어 설명에 적용된다.

   (1) 만약 candidate를 성공적으로 추가할 수 없는 경우 유저 에이전트는 반드시(MUST) 다음 단계를 실행하는 작업을 큐에 넣어야 한다.

      1〉 만약 connection.[[IsClosed]]가 true이면, 이 단계들을 중단한다.

      2〉 새로 생성된 OperationError로 p를 거부(Reject)하고 이 단계들을 중단한다.

   (2) 만약 candidate가 성공적으로 적용되거나, 후보자가 관리적으로 금지된 경우 유저 에이전트는 반드시(MUST) 다음 단계를 실행하는 작업을 큐에 넣어야 한다.

      1〉 만약 connection.[[IsClosed]]이 true이면, 이 단계를 중단한다.

      2〉 만약 connection.[[PendingRemoteDescription]]이 null이 아니고, candidate가 처리 된 ICE 세대를 나타내는 경우, connection.[[PendingRemoteDescription]].sdp에 candidate를 추가한다.

      3〉 만약 connection.[[CurrentRemoteDescription]]이 null이 아니고, candidate가 처리 된 ICE 세대를 나타내는 경우, connection.[[CurrentRemoteDescription]].sdp에 candidate를 추가한다.

      4〉 undefined으로 p를 해결한다.

8) p를 리턴한다.

UA가 이 주소에 대한 연결 시도를 허용하지 않기로 결정한 경우 후보는 관리적으로 금지(administratively prohibited)된다.

개인 정보 보호를 위해, 개발자에게 주소/포트 차단 여부에 대한 표시가 없다; 주소에서 응답이 없는 것과 똑같이 동작한다.

UA는 반드시(MUST) [Fetch] 차단 불량 포트(block bad port) 리스트의 주소에 대한 연결을 금지해야 하며, 다른 주소에 대한 연결을 금지하도록 선택해야(MAY) 한다.

만약 RTCConfiguration의 iceTransportPolicy 멤버가 relay인 경우, mDNS 후보 및 DNS 후보와 같이 외부 확인이 필요한 후보는 반드시(MUST) 금지되어야 한다.

참고

WebIDL 처리로 인해, addIceCandidate(null)는 기본 사전이 있는 호출로 해석되며, 위의 알고리즘에서는 모든 미디어 설명 및 ICE 후보 생성에 대한 후보의 종료를 나타낸다. 이것은 레거시를 위해 설계된 것이다.

restartIce

restartIce 메소드는 RTCPeerConnection에게 ICE를 다시 시작해야 한다는 것을 알려준다. createOffer에 대한 후속 호출은 [ICE]의 섹션 9.1.1.1에 설명한 대로 ICE를 다시 시작하는 설명을 생성한다.

이 메소드가 호출되면, 유저 에이전트는 다음 단계들을 실행해야(MUST) 한다:

1. connection을 메소드가 호출된 RTCPeerConnection으로 설정한다.
2. connection.[[LocalIceCredentialsToReplace]]을 비우고, connection.[[CurrentLocalDescription]]에 있는 모든 ICE 인증 정보([ICE] 섹션 15.4에 정의된 ice-ufrag 및 ice-pwd)와 connection.[[LocalIceCredentialsToReplace]]에 있는 모든 ICE 인증 정보로 채운다.
3. connection에 대해 협상이 필요한 플래그를 업데이트한다.

getConfiguration

이 RTCPeerConnection 객체의 현재 구성을 나타내는 RTCConfiguration 객체를 리턴한다.

이 메소드가 호출되면, 유저 에이전트는 [[Configuration]] 내부 슬롯에 저장된 RTC Configu-

ration 객체를 리턴해야 한다.

setConfiguration

setConfiguration 메소드는 이 RTCPeerConnection 객체의 구성을 업데이트 한다. 여기에는 ICE 에이전트의 구성 변경이 포함된다. [JSEP](섹션 3.5.1.)에서 언급했듯이, ICE 구성이 새로운 수집 단계가 필요한 방식으로 변경되면, ICE를 다시 시작해야 한다.

setConfiguration 메소드가 호출되면, 유저 에이전트는 다음 단계를 반드시(MUST) 실행해야 한다.

1. connection을 메소드가 호출된 RTCPeerConnection로 설정한다.
2. 만약 connection.[[IsClosed]] is true이면, InvalidStateError를 발생(throw)시킨다.
3. configuration에서 지정한 구성을 설정한다.

close

close 메소드가 호출되면, 유저 에이전트는 반드시(MUST) 다음 단계들을 실행해야 한다.

1. connection을 메소드가 호출된 RTCPeerConnection 객체로 지정한다.
2. connection과 false 값으로 연결을 닫는다.

connection 및 disappear 부울(boolean)이 지정된 연결 닫기 알고리즘은 다음과 같다:

1. 만약 connection.[[IsClosed]]이 true이면, 이 단계들을 중단한다.
2. connection.[[IsClosed]]를 true로 설정한다.
3. connection의 시그널링 상태를 “closed”로 설정한다. 이것은 어떤 이벤트도 발생시키지 않는다.
4. transceivers를 CollectTransceivers 알고리즘을 실행한 결과로 지정한다.
   transceivers의 모든 RTCRtpTransceiver transceiver에 대해, 다음 단계를 실행한다:
   1) 만약 transceiver.[[Stopped]]가 true이면, 아래 단계를 중단한다.
   2) transceiver와 disappear로 RTCRtpTransceiver를 중지한다.
5. 각 connection의 RTCDataChannel의 [[ReadyState]] 슬롯을 "closed"로 설정한다.

참고

The RTCDataChannels will be closed abruptly and the closing procedure will not be invoked.
RTCDataChannel가 갑자기 닫히고 종료 절차가 호출되지 않는다.

6. 만약 connection.[[SctpTransport]]이 null이 아니면, SCTP ABORT 청크를 전송하여 기본 SCTP 연결을 해제하고 [[SctpTransportState]]를 "closed"로 설정한다.
7. 각 connection의 RTCDtlsTransports의 [[DtlsTransportState]] 슬롯을 "closed"로 설정한다.
8. connection의 ICE 에이전트를 파괴하고, 활성 ICE 처리를 갑자기 종료하고 모든 관련 리소스 (예. TURN 권한)를 해제한다.
9. 각 connection의 RTCIceTransports의 [[IceTransportState]] 슬롯을 "closed"로 설정한다.
10. connection의 ICE 연결 상태를 "closed"로 설정한다. 이것은 어떤 이벤트도 발생시키지 않는다.
11. Set connection's connection state to "closed". This does not fire any event. connection의 연결 상태를 "closed"로 설정한다. 이벤트가 발생하지 않는다.

③ 레거시 인터페이스 확장

참고

오버로드된 함수는 부분 인터페이스에서 정의할 수 없기 때문에 RTCPeerConnection 인터페이스의 기본 정의에 이러한 메소드의 IDL 정의가 문서화되어 있다.

이 섹션의 메소드를 지원하는 것은 선택 사항이다. 그러나, 이 메소드들이 지원되는 경우 여기에 지정한 내용에 따라 구현해야 한다.

참고

RTCPeerConnection에 있던 addStream메소드는 다음과 같이 폴리필(polyfill)하기 쉽다.

```
RTCPeerConnection.prototype.addStream = function(stream) {
 stream.getTracks().forEach((track) => this.addTrack(track, stream));
};
```

ⓐ 메소드 확장

메소드

createOffer

createOffer 메소드가 호출되면, 사용자 에이전트는 다음 단계를 반드시(MUST) 실행해야 한다:

1. successCallback이 메소드의 첫 번째 인자가 되도록 한다.
2. failureCallback을 메소드의 두 번째 인자가 가리키는 콜백(callback)으로 지정한다.
3. options을 메소드의 세 번째 인자가 가리키는 콜백(callback)으로 지정한다.
4. RTCPeerConnection's createOffer() 메소드에서 options를 단일 인자로 사용하여 지정된 단계를 실행하고, p를 결과 promise로 둔다.
5. p가 offer 값으로 이행되면, offer를 인자로 사용하여 successCallback을 호출한다.
6. r을 이유로 p가 거부되면, r을 인자로 사용해서 failureCallback를 호출한다.
7. undefined로 해결된 promise를 리턴한다.

setLocalDescription

setLocalDescription 메소드가 호출되면, 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야 한다:

1. description를 메소드의 첫 번째 인자로 지정한다.
2. successCallback이 메소드의 두 번째 인자가 가리키는 콜백(callback)으로 지정한다.
3. failureCallback를 메소드의 세 번 째 인자가 가리키는 콜백(callback)으로 지정한다.
4. 유일한 인자로 description을 사용하여 RTCPeerConnection의 setLocalDescription 메소드에 지정된 단계를 실행하고, p를 결과 promise로 둔다.
5. p가 충족되면, undefined를 인자로 사용하여 successCallback을 호출한다.

6. r을 이유로 p가 거부되면, r을 인자로 사용해서 failureCallback을 호출한다.
7. undefined로 해결된 promise를 리턴한다.

createAnswer

참고

레거시 createAnswer 메소드는 알려진 레거시 createAnswer 구현이 지원하지 않았기 때문에 RTCAnswerOptions 매개변수를 사용하지 않는다.

createAnswer 메소드가 호출되면, 사용자 에이전트는 반드시(MUST) 다음 단계를 실행한다:

1. successCallback이 메소드의 첫 번째 인자가 되도록 한다.
2. failureCallback을 메소드의 두 번째 인자의 콜백으로 지정한다.
3. RTCPeerConnection의 createAnswer() 메소드에 지정된 단계를 인자 없이 실행하고, p를 결과 promise로 둔다.
4. answer값으로 p를 이행하면, 응답을 인자로 사용하여 successCallback을 호출한다.
5. r을 이유로 p가 거부되면, r을 인자로 failureCallback을 호출한다.
6. undefined로 해결된 promise를 반환한다.

setRemoteDescription

setRemoteDescription 메소드가 호출되면, 사용자 에이전트는 반드시(MUST) 다음 단계를 실행한다:

1. description이 첫 번째 인자가 되도록 한다.
2. successCallback을 메소드의 두 번째 인자의 콜백으로 지정한다.
3. failureCallback을 메소드의 세 번째 인자의 콜백으로 지정한다.
4. RTCPeerConnection의 setRemoteDescription 메소드에 지정된 단계를 description을 유일한 인자로 사용하여 실행하고, p를 결과 promise로 둔다.
5. p가 충족되면, undefined을 인자로 사용하여 successCallback을 호출한다.
6. r을 근거로 p가 거부되면, r을 인자로 failureCallback을 호출한다.
7. undefined로 해결된 promise를 반환한다.

addIceCandidate

addIceCandidate 메소드가 호출되면, 사용자 에이전트는 반드시(MUST) 다음 단계를 실행한다:

1. candidate를 메소드의 첫 번째 인자로 둔다.
2. successCallback을 메소드의 두 번째 인자의 콜백으로 지정한다.
3. failureCallback을 메소드의 세 번째 인자의 콜백으로 지정한다.
4. RTCPeerConnection의 addIceCandidate() 메소드에 지정된 단계를 candidate을 유일한 인자로 사용하여 실행하고, p를 결과 promise로 둔다.
5. p가 충족되면, undefined을 인자로 사용하여 successCallback을 호출한다.
6. r을 근거로 p가 거부되면, r을 인자로 failureCallback을 호출한다.
7. undefined로 해결된 promise를 반환한다.

콜백 정의(Callback Definitions)

이 콜백(callbacks)들은 레거시 API에서만 사용된다.

RTCPeerConnectionErrorCallback

WebIDL

```
callback RTCPeerConnectionErrorCallback = undefined (DOMException error);
```

RTCPeerConnectionErrorCallback 콜백 인자들

error, DOMException 타입

무엇이 잘못되었는지에 대한 정보를 캡슐화하는 에러 객체이다.

RTCSessionDescriptionCallback

WebIDL

```
callback RTCSessionDescriptionCallback = undefined (RTCSessionDescriptionInit
description);
```

RTCSessionDescriptionCallback 콜백 인자들

description, RTCSessionDescriptionInit 타입

SDP를 담고있는 객체이다.

ⓑ 레거시 구성 확장

이 섹션에서는 RTCPeerConnection에 추가된 미디어 외에 제안(offer) 생성 방법에 영향을 주는데 사용할 수 있는 레거시 확장 세트에 대해 설명한다. 개발자들은 대신 RTCRtpTransceiver API를 사용하는 것을 권장한다.

이 섹션에 지정된 레거시 옵션을 사용해서 createOffer를 호출할 경우, 일반 createOffer 단계 대신 다음 단계를 실행한다:

1. options를 메소드의 첫 번째 인수로 둔다.
2. connection을 현재 RTCPeerConnection 객체로 둔다.
3. 종류가 kind인 options의 각 offerToReceive〈Kind〉 멤버에 대해 다음 단계를 실행한다:
   1) 만약 사전 멤버의 값이 false인 경우,
      (1) 송수신기(transceiver) 종류 kind의 각 논스톱 “sendrecv” 송수신기(transceiver)에 대해 transceiver.[[Direction]]을 “sendonly”로 설정한다.
      (2) 송수신기(transceiver) 종류 kind의 각 논스톱 “recvonly” 송수신기(transceiver)에 대해 transceiver.[[Direction]]을 "inactive"로 설정한다.

      다음 옵션이 있으면 계속한다.
   2) 만약 connection에 송수신기(transceiver) 종류 kind의 중단되지 않은 "sendrecv" 또는 "recvonly" 송수신기(transceiver)가 있는 경우, 다음 옵션이 있으면 계속한다.
   3) 이 작업은 협상에 필요한 플래그를 반드시 업데이트 해서는 안 되는(MUST NOT) 것을 제외하면, 송수신기(transceiver)가 connection.addTransceiver(kind)와 동등한 호출의 결과여야 한다.
   4) 이전 작업에서 에러가 발생해서 transceiver가 설정되지 않은 경우, 이 단계를 중단한다.
   5) transceiver.[[Direction]]를 "recvonly"로 설정한다.
4. Run the steps specified by createOffer to create the offer.
   요청(offer)를 작성하려면 createOffer에서 지정한 단계를 실행한다.

WebIDL

```
partial dictionary RTCOfferOptions {
 boolean offerToReceiveAudio;
 boolean offerToReceiveVideo;
};
```

속성

offerToReceiveAudio, boolean 타입

이 설정은 오디오의 방향성에 대한 추가 제어를 제공한다. 예를 들어, 오디오 전송 여부에 관계없이 오디오를 수신할 수 있는지 확인하는 데 사용할 수 있다.

offerToReceiveVideo, boolean 타입

이 설정은 비디오 방향에 대한 추가 제어를 제공한다. 예를들어, 비디오 전송 여부에 관계없이 비디오를 수신할 수 있는지 확인하는 데 사용할 수 있다.

④ 가비지 콜렉션

RTCPeerConnection 객체는 이벤트가 객체에서 이벤트 핸들러를 트리거 할 수 있는 한 가비지 콜렉션되지 않아야(MUST) 한다. 객체의 [[IsClosed]] 내부 슬롯이 true이면, 이러한 이벤트 핸들러가 트리거되지 않으므로 객체를 가비지 콜렉션하는 것이 안전하다.

RTCPeerConnection에 연결된 모든 RTCDataChannel 및 MediaStreamTrack 객체에는 RTCPeerConnection 객체에 대한 강력한 참조를 가지고 있다.

(5) 오류 처리

① 일반 원칙

promise를 리턴하는 모든 메소드는 promise의 표준 오류 처리 규칙에 따라 관리된다. promise를 반환하지 않는 메소드는 오류를 나타내기 위해 예외를 던질(throw) 수 있다.

(6) 세션 설명 모델(Session Description Model)

① RTCSdpType

RTCSdpType 열겨형은 RTCSessionDescriptionInit, RTCLocalSessionDescriptionInit, 또는 RTCSessionDescription 인스턴스의 타입을 설명한다.

WebIDL

```
enum RTCSdpType {
 "offer",
 "pranswer",
 "answer",
 "rollback"
};
```

열거형	설명
offer	"offer"의 RTCSdpType은 설명(description)이 반드시(MUST) [SDP] 제안(offer)으로 취급해야 함을 나타낸다.
pranswer	"pranswer"의 RTCSdpType은 반드시(MUST) [SDP] 답변(단, 최종 답변이 아닌)으로 취급해야 함을 나타낸다. SDP pranswer로 사용되는 설명은 SDP 제안(offer)에 대한 응답 혹은 이전에 전송된 SDP pranswer에 대한 업데이트로 적용될 수 있다.
answer	“answer”의 RTCSdpType은 반드시(MUST) 설명을 [SDP] 최종 답변으로 취급해야 하며, 제안-답변의 교환이 완료된 것으로 간주되어야 함(MUST)을 나타낸다. SDP 응답으로 사용되는 설명은 SDP 제안에 대한 응답으로 적용되거나 이전에 전송된 SDP pranswer에 대한 업데이트로 적용될 수 있다.
rollback	“rollback”의 RTCSdpType은 반드시(MUST) 설명이 현재 SDP 협상 취소로 간주되어 하며, SDP [SPD] 제안을 이전 안정 상태로 이동하는 것으로 처리되어야 한다. 아직 성공적인 오퍼-답변 협상이 없는 경우 이전 안정 상태의 로컬 또는 원격 SDP 설명이 null일 수 있음을 참고한다. "answer" 혹은 "pranswer"는 롤백할 수 없다.

② RTCSessionDescription 클래스

RTCSessionDescription 클래스는 RTCPeerConnection에서 로컬 및 원격 세션 설명을 노출하는 데 사용된다.

WebIDL

```
[Exposed=Window]
interface RTCSessionDescription {
 constructor(RTCSessionDescriptionInit descriptionInitDict);
 readonly attribute RTCSdpType type;
 readonly attribute DOMString sdp;
 [Default] object toJSON();
};
```

생성자(Constructors)

constructor()

RTCSessionDescription() 생성자는 사전 인자(argument), 설명을 사용하며, 이 내용은 새 RTCSessionDescription 객체를 초기화하는데 사용된다. 이 생성자는 더 이상 사용되지 않으며, 레거시 호환성을 위해서만 존재한다.

속성

type, RTCSdpType 타입, 읽기전용

RTCSessionDescription의 타입

sdp, DOMString 타입, 읽기전용, 기본값은 ""

SDP [SDP]의 문자열 표현

메소드

toJSON()

호출되면, [WEBIDL]의 기본 toJSON 단계를 실행.

WebIDL

```
dictionary RTCSessionDescriptionInit {
 required RTCSdpType type;
 DOMString sdp = "";
};
```

RTCSessionDescriptionInit 사전 멤버

type, RTCSdpType 타입, 필수

이 설명(description)의 타입.

sdp, DOMString 타입

SDP [SDP]의 문자열 표현; 만약 type이 "rollback"이면, 이 멤버는 사용되지 않음.

WebIDL

```
dictionary RTCLocalSessionDescriptionInit {
 RTCSdpType type;
 DOMString sdp = "";
};
```

RTCLocalSessionDescriptionInit 사전 멤버

type, RTCSdpType 타입

이 설명(description)의 타입. 존재하지 않는 경우, setLocalDescription은 RTCPeerConnection의 시그날링 상태를 기반으로 타입을 유추.

sdp, DOMString 타입

SDP [SDP]의 문자열 표현; 만약 type 이 "rollback"이면, 이 멤버는 사용되지 않음.

(7) 세션 협상 모델

RTCPeerConnection의 상태를 많은 변경들은 원하는 효과를 얻기 위해 시그널링 채널을 통해 원격 부분과 통신해야 한다. 앱은 negotiationneeded 이벤트를 수신하여 언제 시그널링 해야 하는지에 대한 정보를 유지할 수 있다. 이 이벤트는 [[NegotiationNeeded]] 내부 슬롯으로 표시되는 연결의 협상 필요 플래그(negotiation-needed flag) 상태에 따라 발생된다.

① 협상 필요 설정(Setting Negotiation-Needed)

이 섹션은 비표준이다.

만약 시그널링이 필요한 RTCPeerConnection에서 작업이 수행되면, 해당 연결은 협상이 필요한 것으로 표시된다. 이러한 작업의 예로는 RTCRtpTransceiver 추가 또는 중지, 또는 첫 번째 RTCDataChannel 추가가 있다.

구현 내의 내부 변경은 협상이 필요한 것으로 표시된 연결 내에서도 발생할 수 있다.

협상이 필요한 플래그를 업데이트하기 위한 정확한 절차는 아래에 명시되어 있다.

② 협상-필요 청산(Clearing Negotiation-Needed)

이 섹션은 비표준이다.

협상-필요 플래그(negotiation-needed flag)는 "answer" 타입의 RTCSessionDescription이 적용되고, 제공된 설명이 현재 RTCPeerConnection에 존재하는 RTCRtpTransceiver 및 RTCDataChannel의 상태와 일치하면 지워진다. 특히, 모든 non-stopped 트랜시버의 로컬 설명에 속성이 일치하는 관련 섹션이 있으며, 데이터 채널이 생성된 경우 로컬 설명에 데이터 섹션이 있음을 의미한다.

협상-필요 플래그(negotiation-needed flag)를 업데이트하기 위한 정확한 절차는 아래에 지정되어 있다.

③ 협상 필요 플래그 업데이트 (Updating the Negotiation-Needed flag)

아래 프로세스는 이 문서의 다른 부분에서 참조되는 곳에서 발생한다. 협상에 영향을 미치는 구현 내 내부 변경의 결과로 발생할 수도 있다. 이러한 변경이 발생하면 유저 에이전트는 협상-필요 플래그(negotiation-needed flag)를 반드시(MUST) 업데이트 해야 한다.

연결을 위한 협상-필요 플래그 업데이트를 하려면 다음 단계를 실행한다:

1. 만약 connection.[[Operations]]의 길이가 0이 아니면, connection.[[UpdateNegotiationNeededFlagOnEmptyChain]]을 true로 설정하고, 이 단계를 중단한다.
2. 다음 단계를 실행하기 위해 태스크를 큐에 넣는다:
    1) 만약 connection.[[IsClosed]]가 true이면, 이 단계를 중단한다.
    2) 만약 connection.[[Operations]]가 0이 아니면, connection.[[UpdateNegotiationNeededFlagOnEmptyChain]]을 true로 설정하고, 이 단계를 중단한다.
    3) 만약 connection의 시그널링 상태가 "stable"이 아닌 경우, 이 단계를 중단한다.

참고

협상-필요 플래그(negotiation-needed flag)는 RTCSessionDescription 설정 단계의 일부로, 상태가 "stable"으로 전환되면 업데이트된다.

    4) 만약 협상 필요 확인 결과가 false이면, connection.[[NegotiationNeeded]]의 false 설정으로 협상-필요 플래그(negotiation-needed flag) 삭제를 하고, 이 단계를 중단한다.
    5) 만약 connection.[[NegotiationNeeded]]가 이미 true이면, 이 단계를 중단한다.
    6) connection.[[NegotiationNeeded]]을 true로 설정한다.
    7) connection에서 negotiationneeded 라는 이름의 이벤트를 시작한다.

참고

작업 대기열은 연결에 대한 여러 수정이 한번에 수행되는 일반적인 상황에서 negotiationneeded가 조기에 실행되는 것을 방지한다.
또한, 운영 체인이 비어 있을 때 negotiationneeded 실행에 의한 협상 방법으로 경주(racing)를 피한다.

connection에 협상이 필요한지 확인하려면, 다음 검사를 수행한다:

1. 이 섹션의 시작 부분에 설명한 대로 만약 구현별 협상이 필요한 경우, true를 리턴한다.
2. 만약 connection.[[LocalIceCredentialsToReplace]]가 비어 있지 않으면, true를 리턴한다.

3. description을 connection.[[CurrentLocalDescription]]으로 지정한다.
4. 만약 connection이 RTCDataChannel을 생성하고, description의 m= 섹션이 아직 데이터에 대해 협상되지 않은 경우, true를 리턴한다.
5. connection의 트랜시버 세트에 있는 각 transceiver에 대해 다음 검사를 수행한다:
   1) 만약 transceiver.[[Stopping]]이 true이고 transceiver.[[Stopped]]가 false이면, true를 리턴한다.
   2) 만약 transceiver가 중지되지 않았고 description의 m= 섹션과 아직 연결되지 않은 경우, true를 리턴한다.
   3) 만약 transceiver가 stopped되지 않고 description의 m= 섹션과 연결되어 있으면, 다음 검사를 수행한다:
      (1) 만약 transceiver.[[Direction]]이 "sendrecv" 또는 "sendonly"이고, description의 관련 m= 섹션에 단일 a=msid 라인, 또는 a=msid 라인의 MSID 수가 포함되지 않은 경우 m= 섹션에서, 또는 MSID 값 자체가 transceiver.sender.[[AssociatedMediaStreamIds]]에 있는 것과 다르면, true를 리턴한다.
      (2) 만약 description이 "offer" 유형이고, connection.[[CurrentLocalDescription]] 또는 connection.[[CurrentRemoteDescription]] 모두에서 연관된 m= 섹션의 방향이 transceiver.[[Direction]]과 일치하지 않으면, true를 리턴한다. 이 단계에서, 방향을 [[CurrentRemoteDescription]]에서 찾은 방향과 비교할 때, 설명의 방향을 반대로 하여 상대방의 관점을 표현해야 한다.
      (3) 만약 description이 "answer" 타입이고, description에서 연관된 m= 섹션의 방향과 교차하는 transceiver.[[Direction]]과 일치하지 않는 경우([JSEP](섹션 5.3.1에 설명), true를 리턴한다.
      (4) 만약 transceiver가 stopped 되고 m= 섹션과 연결되어 있지만, 연결된 m= 섹션이 connection.[[CurrentLocalDescription]] 또는 connection.[[CurrentRemoteDescription]]에서 아직 거부되지 않은 경우, true를 리턴한다.
6. 만약 이전의 모든 검사가 수행되고 true가 리턴되지 않았다면, 협상 할 것은 아무것도 남지 않았다: false를 리턴한다.

(8) 대화형 연결 설정을 위한 인터페이스

① RTCIceCandidate 인터페이스

이 인터페이스는 [ICE] 섹션 2에 설명된 ICE 후보를 설명한다. candidate, sdpMid, sdpMLineIndex 및 usernameFragment를 제외한 나머지 속성은 잘 구성된 경우 candidateInitDict에서 candidate 멤버를 구문 분석해서 파생된다.

WebIDL

```
[Exposed=Window]
interface RTCIceCandidate {
 constructor(optional RTCIceCandidateInit candidateInitDict = {});
 readonly attribute DOMString candidate;
 readonly attribute DOMString? sdpMid;
 readonly attribute unsigned short? sdpMLineIndex;
 readonly attribute DOMString? foundation;
 readonly attribute RTCIceComponent? component;
 readonly attribute unsigned long? priority;
 readonly attribute DOMString? address;
 readonly attribute RTCIceProtocol? protocol;
 readonly attribute unsigned short? port;
 readonly attribute RTCIceCandidateType? type;
 readonly attribute RTCIceTcpCandidateType? tcpType;
 readonly attribute DOMString? relatedAddress;
 readonly attribute unsigned short? relatedPort;
 readonly attribute DOMString? usernameFragment;
 RTCIceCandidateInit toJSON();
};
```

생성자

constructor()

RTCIceCandidate() 생성자는 사전 인자인 candidateInitDict을 사용하는데, 이 인자는 새로운 RTCIceCandidate 객체를 초기화하는 데 콘텐츠가 사용된다.

호출이 되면, 다음 단계를 실행한다:

1. candidateInitDict의 sdpMid와 sdpMLineIndex 멤버가 모두 null이면, TypeError를 발생(throw)시킨다.
2. candidateInitDict로 RTCIceCandidate를 만든 결과를 리턴한다.

candidateInitDict 사전으로 RTCIceCandidate를 생성하려면, 다음 단계를 실행한다:

1. iceCandidate 를 새로 생성된 RTCIceCandidate 객체로 둔다.
2. null로 초기화된, iceCandidate의 속성에 대한 내부 슬롯(internal slots)을 만든다: foundation, component, priority, address, protocol, port, type, tcpType, related Address 및 relatedPort.
3. candidateInitDict 에서 이름을 따서 초기화된 iceCandidate의 다음 속성에 대한 내부 슬롯(internal slots)을 만든다: candidate, sdpMid, sdpMLineIndex, username Fragment
4. candidate를 candidateInitDict 의 candidate 사전 멤버로 지정한다. candidate가 빈 문자열이 아닌 경우, 다음 단계를 실행한다:
   1) candidate-attribute 문법을 사용해서 candidate를 분석(parse)한다.
   2) 만약 candidate-attribute 분석(parse)에 실패한 경우, 이 단계를 중단한다.
   3) 만약 분석(parse) 결과 필드가 iceCandidate의 해당 속성에 대해 유효하지 않은 값을 나타내는 경우, 이 단계를 중단한다.
   4) iceCandidate의 해당 내부 슬롯(internal slots)을 분석(parse)된 결과의 필드 값으로 설정한다.
5. iceCandidate를 리턴한다.

참고

RTCIceCandidate의 생성자는 candidateInitDict의 사전 멤버에 대한 기본 파싱 및 유형 체크만 수행한다. RTCIceCandidate 객체를 addIceCandidate()에 전달할 때 해당 세션 설명과 함께 candidate, sdpMid, sdpMLineIndex, usernameFragment의 올바른 형식에 대한 자세한 유효성 검사가 수행된다.
이전 버전과의 호환성을 유지하기 위해 후보 속성 파싱 오류는 무시된다. 이 경우 candidate 속성은 candidateInitDict에 제공된 원시 candidate 문자열을 보유하지만 foundation, priority 등과 같은 파생 속성은 null로 설정된다.

속성

아래의 대부분 속성은 [ICE] 섹션 15.1에 정의되어 있다.

candidate, DOMString 타입, 읽기전용

이 속성은 [ICE] 섹션 15.1에 정의 된 바와 같이 candidate-attribute(후보-속성)을 전달한다. 이 RTCIceCandidate가 최종 후보 표시 또는 피어 반사 원격 후보를 나타내는 경우, candidate는 빈 문자열이다.

sdpMid, DOMString 타입, 읽기전용, null 허용

만약 null이 아닌 경우, 이 후보와 관련된 미디어 컴포넌트에 대해 [RFC5888]에 정의된 미디어 스트림 “식별-태그(identification-tag)"가 포함된다.

sdpMLineIndex, unsigned short 타입, 읽기전용, null 허용

null이 아닌 경우, 후보가 연관된 SDP 미디어 설명의 인덱스(0부터 시작)를 나타낸다.

foundation, DOMString 타입, 읽기전용, null 허용

ICE가 여러 RTCIceTransports에 나타나는 후보를 상호 연관시킬 수 있는 고유 식별자이다.

component, RTCIceComponent 타입, 읽기전용, null 허용

후보의 할당된 네트워크 구성요소이다("rtp" 또는 "rtcp"). 이는 component-id에 정의대로 문자열 표현으로 디코딩된 후보 속성의 component-id 필드에 해당된다.

priority, unsigned long 타입, 읽기전용, null 허용

후보에 할당된 우선순위이다.

address, DOMString 타입, 읽기전용, null 허용

IPv4 주소, IPv6 주소 및 FQDN(fully qualified domain names)을 허용하는 후보 주소이다. 이는 candidate-attribute의 connection-address필드에 해당한다.

원격 후보는 예를 들어 [[SelectedCandidatePair]].remote를 통해 노출될 수 있다. 기본적으로 유저 에이전트는 노출된 원격 후보에 대해 address 속성을 null로 남겨 두어야 한다. RTCPeerConnection 인스턴스가 addIceCandidate를 사용하여 웹 애플리케이션에서 주소를 학습하면, 유지 에이전트는 새로 학습된 주소로 원격 후보를 나타내는 RTCPeerConnection 인스턴스의 모든 RTCIceCandidate에 주소 속성 값을 노출할 수 있다.

참고

ICE를 통해 수집되고 RTCIceCandidate 인스턴스의 애플리케이션에 가시적으로 만들어진 후보자에게 노출되는 주소는 사용자가 WebRTC를 지원하지 않는 브라우저에서 예상보다 더 많은 장치와 사용자에 대한 정보(예. 위치, 로컬 네트워크 토폴로지)를 노출할 수 있다. 이러한 주소는 항상 애플리케이션에 노출되고, 통신 당사자에게 잠재적으로 노출되며, 특정 사용자 동의없이 노출될 수 있다(예. 데이터 채널과 함께 사용되는 피어 연결, 또는 미디어 수신 전용).

이러한 주소는 일시적 또는 영구적인 상호출처 상태로도 사용될 수 있으므로, 장치의 지문 채취에 기여한다.

애플리케이션은 ICE 에이전트가 RTCConfiguration의 iceTransportPolicy 멤버를 통해 릴레이 후보만 보고 하도록 강제함으로써 일시적 또는 영구적으로 통신 당사자에게 주소가 노출되는 것을 방지할 수 있다.

애플리케이션 자체에 노출되는 주소를 제한하기 위해, 브라우저는 [RTCWEB-IP-HANDLING]에 정의된 바와 같이 로컬 주소 공유와 관련된 다양한 정책을 유저에게 제공할 수 있다.

protocol, RTCIceProtocol 타입, 읽기전용, null 허용

후보의 프로토콜이다("udp"/"tcp"). 이는 candidate-attribute의 transport 필드에 해당된다.

port, unsigned short 타입, 읽기전용, null 허용

The port of the candidate.

후보의 포트이다.

type, RTCIceCandidateType 타입, 읽기전용, null 허용

후보자의 타입이다. 이는 candidate-attribute의 candidate-types 필드에 해당한다.

tcpType, RTCIceTcpCandidateType 타입, 읽기전용, null 허용

만약 protocol이 "tcp"이면, tcpType은 TCP 후보의 타입을 나타낸다. 그렇지 않으면, tcpType은 null이다. 이것은 candidate-attribute의 tcp-type 필드에 해당한다.

relatedAddress, DOMString 타입, 읽기전용, null 허용

릴레이 또는 반사 후보와 같이 다른 후보에서 파생된 후보의 경우, relatedAddress는 파생된 후보의 IP 주소이다. 호스트 후보의 경우 relatedAddress는 null이다. 이는 candidate-attribute의 rel-address 필드에 해당한다.

relatedPort, unsigned short 타입, 읽기전용, null 허용

릴레이 또는 반사 후보와 같이 다른 후보에서 파생 된 후보의 경우 relatedPort는 파생된 후보의 포트이다. 호스트 후보의 경우 relatedPort는 null이다. 이는 candidate-attribute의 rel-port 필드에 해당한다.

usernameFragment, DOMString 타입, 읽기전용, null 허용

This carries the ufrag as defined in section 15.4 of [ICE].

이것은 [ICE] 섹션 15.4에 정의된 ufrag를 전달한다.

메소드

toJSON()

RTCIceCandidate 인터페이스의 toJSON() 작업을 호출하려면, 다음 단계를 실행한다:

1. json을 새로운 RTCIceCandidateInit 사전이 되도록 한다.

2. «candidate, sdpMid, sdpMLineIndex, usernameFragment»의 각 속성 식별자 attr에 대해:
    1) 이 RTCIceCandidate 객체가 주어지면, value를 attr로 식별되는 속성의 기본 값을 가져온 결과라고 가정한다.
    2) json[attr]을 값으로 설정한다.
3. json을 리턴한다.

WebIDL

```
dictionary RTCIceCandidateInit {
 DOMString candidate = "";
 DOMString? sdpMid = null;
 unsigned short? sdpMLineIndex = null;
 DOMString? usernameFragment = null;
};
```

RTCIceCandidateInit 사전 멤버

candidate, DOMString 타입, 기본값은 ""

이것은 [ICE] 섹션 15.1에 정의된 후보 속성을 전달한다. 이것이 후보의 끝(end-of- candidates) 표시를 나타내는 경우, candidate는 빈 문자열이다.

sdpMid, DOMString 타입, null 허용, 기본값은 null

null이 아닌 경우, 이 후보가 연결된 미디어 구성 요소에 대해 [RFC5888]에 정의된 미디어 스트림 "identification-tag"를 포함한다.

sdpMLineIndex, unsigned short 타입, null 허용, 기본값은 null

만약null이 아닌 경우, 이 후보가 연결된 SDP에 있는 미디어 설명의 인덱스 (0에서 시작)를 나타낸다.

usernameFragment, DOMString 타입, null 허용, 기본값은 null

만약 null이 아닌 경우, [ICE] 섹션 15.4 정의 된대로 ufrag를 전달한다.

ⓐ candidate-attribute 문법

candidate-attribute 문법은 RTCIceCandidate() 생성자에서 candidateInitDict의 candidate 멤버를 분석하는 데 사용된다.

candidate-attribute의 주 문법은 [ICE] 섹션 15.1에 정의되어 있다. 또한, 브라우저는 [RFC6544] 섹션 4.5에 정의된 바와 같이 ICE TCP에 대한 문법 확장을 반드시 지원해야 한다.

브라우저는 다른 RFC에 정의된 candidate-attribute에 대해 다른 문법 확장을 지원할 수 있다.

ⓑ RTCIceProtocol 열거형

RTCIceProtocol은 ICE 후보(candidate)의 프로토콜을 나타낸다.

WebIDL

```
enum RTCIceProtocol {
 "udp",
 "tcp"
};
```

열거형	설명
udp	A UDP candidate, as described in [ICE]. UDP 후보, [ICE]에 설명
tcp	A TCP candidate, as described in [RFC6544]. TCP 후보, [RFC6544]에 설명

ⓒ RTCIceTcpCandidateType 열거형

RTCIceTcpCandidateType은 [RFC6544]에 정의된 바와 같이 ICE TCP 후보의 유형을 나타낸다.

WebIDL

```
enum RTCIceTcpCandidateType {
 "active",
 "passive",
 "so"
};
```

열거형 설명	
active	“active” TCP 후보는 전송이 아웃 바운드 연결을 열려고 시도는 하지만 들어오는 연결 요청을 받지는 않는 후보
passive	"passive" TCP 후보는 전송이 들어오는 연결 시도는 받을 수 있지만 연결을 시도하지는 않는 후보
so	"so" 후보는 전송이 피어와 동시에 연결을 열려고 시도하는 후보

참고

사용자 에이전트 일반적으로 active ICE TCP 후보군만 수집한다.

ⓓ RTCIceCandidateType 열거형

RTCIceCandidateType은 [ICE] 섹션 15.1에 정의된 바와 같이 ICE 후보의 유형을 나타낸다.

WebIDL

```
enum RTCIceCandidateType {
 "host",
 "srflx",
 "prflx",
 "relay"
};
```

열거형 설명	
host	[ICE] 섹션 4.1.1.1에 정의된 호스트 후보.
srflx	[ICE] 섹션 4.1.1.2에 정의된 서버 재귀(reflexive) 후보.
prflx	[ICE] 섹션 4.1.1.2에 정의된 피어 재귀(reflexive) 후보.
relay	[ICE] 섹션 7.1.3.2.1에 정의된 릴레이 후보.

② RTCPeerConnectionIceEvent

RTCPeerConnection의 icecandidate 이벤트는 RTCPeerConnectionIceEvent 인터페이스를 사용한다.

RTCIceCandidate 객체를 포함하는 RTCPeerConnectionIceEvent 이벤트를 발생시킬 때, sdpMid와 sdpMLineIndex 모두에 대한 값을 반드시 포함해야 한다. RTCIceCandidate가 "srflx" 타입 혹은 "relay" 타임인 경우, 이벤트의 url 속성은 후보가 획득된 ICE 서버의 URL로 반드시 설정되어야 한다.

참고

icecandidate 이벤트는 세 가지 다른 유형의 표시에 사용된다.

- 후보자가 모였다. 이벤트의 candidate 멤버는 정상적으로 채워진다. 원격 피어에 신호를 보내고 addIceCandidate로 전달되어야 한다.
- RTCIceTransport는 한 세대의 후보 수집을 완료했으며, [TRICKLE-ICE]의 섹션 8.2에 정의된 후보 종료 표시를 제공하고 있다. 이것은 candidate.candidate가 빈 문자열로 설정됨으로 표시된다. candidate 객체는 원격 피어에 신호를 보내서 일반 ICE 후보처럼 addIceCandidate에 전달해야 하며, 원격 피어에 후보 종료 표시를 제공해야 한다.
- 모든 RTCIceTransports가 후보 수집을 완료했으며, RTCPeerConnection의 RTCIceGatheringState가 "complete"로 전환되었다. 이는 이벤트의 candidate 멤버가 null로 설정되어 표시된다. 이는 이전 버전과의 호환성을 위해서만 존재하며, 이 이벤트는 원격 피어에 신호를 보낼 필요가 없다. 이는 "complete" 상태를 가진 icegatheringstatechange 이벤트와 동일하다.

WebIDL

```
[Exposed=Window]
interface RTCPeerConnectionIceEvent : Event {
 constructor(DOMString type, optional RTCPeerConnectionIceEventInit
eventInitDict = {});
 readonly attribute RTCIceCandidate? candidate;
 readonly attribute DOMString? url;
};
```

생성자

RTCPeerConnectionIceEvent.constructor()

속성

candidate, RTCIceCandidate 타입, 읽기전용, null 허용

candidate 속성은 이벤트를 유발한 새 ICE 후보가 있는 RTCIceCandidate 객체이다. 이 속성은 후보 수집의 끝을 표시하기 위해 이벤트가 생성될 때 null로 설정된다.

참고

여러 미디어 구성 요소가 있는 경우에도 null 후보를 포함하는 하나의 이벤트만 시작된다.

url, DOMString 타입, 읽기전용, null 허용

url 속성은 후보를 수집하는데 사용되는 STUN 또는 TURN 서버를 식별하는 STUN 또는 TURN URL이다. 후보가 STUN 또는 TURN 서버에서 수집되지 않은 경우 이 매개 변수는 null로 설정된다.

WebIDL

```
dictionary RTCPeerConnectionIceEventInit : EventInit {
 RTCIceCandidate? candidate;
 DOMString? url;
};
```

Dictionary RTCPeerConnectionIceEventInit 사전 멤버들

candidate, RTCIceCandidate 타입, null 허용

RTCPeerConnectionIceEvent 인터페이스의 candidate 속성을 참고하라.

url, DOMString 타입, null 허용

url 속성은 이 후보를 수집하는 데 사용되는 STURN 또는 TURN 서버를 식별하는 STUN 또는 TURN URL이다.

③ RTCPeerConnectionIceErrorEvent

RTCPeerConnection의 icecandidateerror 이벤트는 RTCPeerConnectionIceErrorEvent 인터페이스를 사용한다.

WebIDL

```
[Exposed=Window]
interface RTCPeerConnectionIceErrorEvent : Event {
 constructor(DOMString type, RTCPeerConnectionIceErrorEventInit eventInitDict);
 readonly attribute DOMString? address;
 readonly attribute unsigned short? port;
 readonly attribute DOMString url;
 readonly attribute unsigned short errorCode;
 readonly attribute USVString errorText;
};
```

생성자

RTCPeerConnectionIceErrorEvent.constructor()

속성

address, DOMString 타입, 읽기전용, null 허용

address 속성은 STUN 또는 TURN 서버와 통신하는 데 사용되는 로컬 IP 주소이다.

멀티홈 시스템에서, 여러 인터페이스를 사용하여 서버에 연결할 수 있으며, 이 속성을 사용하면 응용 프로그램에서 오류가 발생한 것을 파악할 수 있다.

만약 로컬 IP 주소 값이 로컬 후보의 일부로 아직 노출되지 않은 경우 address 속성은 null로 설정된다.

port, unsigned short 타입, 읽기전용, null 허용

port 속성은 STUN 또는 TURN 서버와 통신하는 데 사용되는 포트이다.

만약 address 속성이 null이면, port 속성도 null로 설정된다.

url, DOMString 타입, 읽기전용

url 속성은 장애가 발생한 STUN 또는 TURN 서버를 식별하는 STUN 또는 TURN URL이다.

errorCode, unsigned short 타입, 읽기전용

errorCode 속성은 STUN 또는 TURN 서버에서 리턴한 숫자 STUN 에러 코드이다 [STUN-PARAMETERS].

만약 호스트 후보가 서버에 도달 할 수 없는 경우, errorCode는 STUN 에러 코드 범위를 벗어난 값 701으로 설정된다. 이 에러는 RTCIceGatheringState가 "gathering"인 동안 서버 URL 당 한 번만 발생한다.

errorText, USVString 타입, 읽기전용

errorText 속성은 STUN 또는 TURN 서버에서 리턴한 STUN 근거(reason) 텍스트이다 [STUN-PARAMETERS].

만약 서버에 연결할 수 없는 경우, errorText는 에러에 대한 세부 정보를 제공하는 구현 별(implementation-specific) 값으로 설정된다.

WebIDL

```
dictionary RTCPeerConnectionIceErrorEventInit : EventInit {
 DOMString? address;
 unsigned short? port;
 DOMString url;
 required unsigned short errorCode;
 USVString statusText;
};
```

RTCPeerConnectionIceErrorEventInit 사전 멤버들

address, DOMString 타입, null 허용

STUN 또는 TURN 서버와 통신하는 데 사용되는 로컬 주소 또는 null이다.

port, unsigned short 타입, null 허용

STUN 또는 TURN 서버와 통신하는 데 사용되는 로컬 포트 또는 null이다.

url, DOMString 타입

장애가 발생한 STUN 또는 TURN 서버를 식별하는 STUN 또는 TURN URL이다.

errorCode, unsigned short 타입, 필수

STUN 또는 TURN 서버에서 리턴한 숫자 STUN 에러 코드이다.

statusText, USVString 타입

STUN 또는 TURN 서버에서 리턴한 STUN 근거(reason) 텍스트이다.

(9) 인증서 관리

RTCPeerConnection 인스턴스가 피어 인증에 사용하는 인증서는 RTCCertificate 인터페이스를 사용한다. 이러한 객체는 generateCertificate 메소드를 사용해서 애플리케이션에서 명시적으로 생성할 수 있으며 새로운 RTCPeerConnection 인스턴스를 구성할 때 RTCConfiguration로 제공할 수 있다.

여기에 제공된 명시적 인증서 관리 기능은 선택 사항이다. 애플리케이션이 RTCPeerConnection을 구성할 때 certificates 구성 옵션을 제공하지 않으면 새 인증서 세트는 사용자 에이전트에 의해 반드시 생성되어야 한다. 해당 세트는 반드시 P-256 곡선의 개인 키가 있는 ECDSA 인증서와 SHA-256 해시가 있는 서명이 반드시 포함되어야 한다.

WebIDL

```
partial interface RTCPeerConnection {
 static Promise<RTCCertificate>
   generateCertificate(AlgorithmIdentifier keygenAlgorithm);
};
```

메소드

generateCertificate, 정적(static)

generateCertificate 함수는 사용자 에이전트가 X.509 인증서[X509V3]와 해당 개인 키를 생성하게 한다. 정보 핸들은 RTCCertificate 인터페이스의 형태로 제공된다. 반환된 RTCCertificate는 PTCPeerConnection에 의해 설정된 DTLS 세션에서 제공되는 인증서를 제어하는 데 사용할 수 있다.

keygenAlgorithm 인수는 인증서와 연결된 개인 키가 생성되는 방식을 제어하는 데 사용된다. keygenAlgorithm 인수는 WebCrypto[WebCryptoAPI] AlgorithmIdentifier 타입을 사용한다.

다음 값은 사용자 에이전트에서 반드시 지원되어야 한다: { name: "RSASSA-PKCS1-v1_5", modulusLength: 2048, publicExponent: new Uint8Array([1, 0, 1]), hash: "SHA-256" }, 및 { name: "ECDSA", namedCurve: "P-256" }.

참고

사용자 에이전트는 수용할 수 있는 작은 값 혹은 고정된 값 세트를 가질 것으로 예상된다.

이 프로세스에서 생성된 인증서에는 서명도 포함되어 있다. 이 서명의 유효성은 호환성에만 관련된다. 공개 키와 결과 인증서 핑거프린트만 RTCPeerConnection에서 사용되지만, 인증서가 제대로 구성된 경우 허용될 가능성이 더 높다. 브라우저는 인증서 서명에 사용하는 알고리즘을 선택합니다; 브라우저는 해시 알고리즘이 필요한 경우 반드시 SHA-256 [FIPS-180-4]를 선택해야 한다.

결과 인증서에는 사용자 혹은 사용자 에이전트에 연결할 수 있는 정보가 절대로 포함되지 않아야 한다. 고유 이름 및 일련 번호에 대한 무작위 값이 사용되어야 한다.

메소드가 호출되면, 사용자 에이전트는 다음 단계를 반드시(MUST) 실행해야 한다:

1. keygenAlgorithm을 generateCertificate의 첫 번째 인수로 지정한다.
2. expires는 592000000의 DOMTimeStamp 값으로 지정한다.

참고

이는 인증서가 기본적으로 generateCertificate 호출 시점으로 부터 30일 후에 만료됨을 의미한다.

3. 만약 keygenAlgorithm이 객체인 경우, 다음 단계를 실행한다:
   1) certificateExpiration은 keygenAlgorithm으로 표현되는 ECMAScript 객체를 RTC CertificateExpiration 사전으로 변환한 결과물로 간주한다.
   2) 만약 오류로 인해 변환을 실패하면, 오류로 거부된 promise를 반환한다.
   3) 만약 certificateExpiration.expires가 undefined가 아닌 경우, expires를 certificateExpiration.expires로 설정한다.
   4) 만약 expires가 31536000000보다 크면, expires를 31536000000로 설정한다.

참고

이는 인증서가 generateCertificate 호출 시점으로 부터 265일 이상 유효할 수 없음을 의미한다.

사용자 에이전트는 expires 값을 추가로 제한할 수 있다.

4. normalizedKeygenAlgorithm을 generateKey의 작업 이름과 RTCPeerConnection에 대한 인증서 생성에 특정한 supportedAlgorithms 값으로 알고리즘을 정규화 한 결과로 간주한다.
5. 만약 위의 정규화 단계가 오류와 함께 실패하면, 오류로 거부된 promise를 반환한다.
6. 만약 normalizedKeygenAlgorithm 인자가 유저 에이전트가 RTCPeerConnection에 대한 인증서를 생성하는데 사용할 수 없거나 사용할 수 없는 알고리즘을 식별하는 경우, NotSupportedError 타입의 DOMException으로 거부된 promise를 리턴한다. 특히, normalizedKeygenAlgorithm은 DTLS 연결을 인증하는 데 사용되는 서명을 생성하는 데 사용할 수 있는 반드시 비대칭 알고리즘이어야 한다.
7. p를 새로운 promise로 둔다.
8. 다음 단계를 병렬로 실행한다:
   1) keygenAlgorithm을 사용해서 normalizedKeygenAlgorithm에서 지정한 키 생성 작업을 수행한다.
   2) generatedKeyingMaterial과 generatedKeyCertificate를 위 단계에서 생성된 개인 키 자료와 인증서로 지정한다.
   3) certificate를 새 RTCCertificate 객체로 지정한다.
   4) certificate.[[Expires]]를 현재 시각에 expires 값을 더해서 설정한다.
   5) certificate.[[Origin]]을 현재 설정 객체의 출처로 설정한다.
   6) 보안 모듈에 generatedKeyingMaterial을 저장하고, handle을 참조 식별자로 둔다.
   7) certificate.[[KeyingMaterialHandle]]을 handle로 설정한다.
   8) certificate.[[Certificate]]를 generatedCertificate로 설정한다.
   9) certificate로 p를 해결한다.
9. p를 리턴한다.

① RTCCertificateExpiration 사전

RTCCertificateExpiration은 generateCertificate에 의해 생성된 인증서의 만료 날짜를 설정하는 데 사용된다.

WebIDL

```
dictionary RTCCertificateExpiration {
 [EnforceRange] DOMTimeStamp expires;
};
```

expires, DOMTimeStamp 타입

선택적인 expires 속성은 generateCertificate로 전달되는 알고리즘의 정의에 추가될 수 있다. 이 매개 변수가 존재하는 경우 현재 시간을 기준으로 RTCCertificate가 유효한 최대 시간을 나타낸다.

② RTCCertificate 인터페이스

RTCCertificate 인터페이스는 WebRTC 통신을 인증하는 데 사용되는 인증서를 나타낸다. 가시적인 특성 외에도, 내부 슬롯에는 생성된 개인 키 정보([[KeyingMaterialHandle]])에 대한 핸들, RTCPeerConnection이 피어와 인증하는 데 사용하는 인증서 ([[Certificate]]) 및 원본 ([[Origin]]) 객체를 생성했다.

WebIDL

```
[Exposed=Window, Serializable]
interface RTCCertificate {
 readonly attribute DOMTimeStamp expires;
 sequence<RTCDtlsFingerprint> getFingerprints();
};
```

속성

expires, DOMTimeStamp 타입, 읽기전용

expires 속성은 1970-01-01T00:00:00Z를 기준으로 밀리초 기반의 날짜 및 시간을 표시하며 해당 값 이후의 인증서는 브라우저에서 유효하지 않은 것으로 간주된다.

지정된 시간이 지난 후 이 인증서를 사용하여 RTCPeerConnection을 구성하려고 하면 실패한다. 참고로, 이 값은 인증서 자체의 notAfter 매개 변수에 반영되지 않을 수 있다.

메소드

getFingerprints

인증서 핑거프린트 리스트를 반환하며, 그 중 하나는 인증 서명에 사용된 다이제스트 알고리즘으로 계산된다.

이 API의 목적상, [[Certificate]] 슬롯은 구조화되지 않은 바이너리 데이터를 포함한다. [[KeyingMaterialHandle]] 내부 슬롯 또는 참조하는 키 자료에 접근하기 위한 메커니즘이 애플리케이션에 제공되지 않는다. 구현은 [[KeyingMaterialHandle]]에서 참조하는 키 자료를 보존하는 방식으로 영구 저장소에서 RTCCertificate 객체를 저장하고 검색하는 애플리케이션을 반드시 지원해야 한다. 구현은 민감한 키 자료를 동일한 프로세스 메모리 공격으로 부터 안전한 보안 모듈에 저장하도록 해야 한다. 이를 통해 개인 키를 저장하고 사용할 수 있지만, 메모리 공격으로 쉽게 읽을 수는 없다.

RTCCertificate 객체는 직렬화 가능한 객체이다[HTML]. 주어진 값과 직렬화된 직렬화 단계는 다음과 같다:

1. serialized.[[Expires]]를 value.expires 속성 값으로 설정한다.
2. serialized.[[Certificate]]를 value.[[Certificate]]에 있는 구조화 되지 않은 바이너리 데이터의 복사본으로 설정한다.
3. serialized.[[Origin]]을 value.[[Origin]]에 있는 구조화되지 않은 바이너리 데이터의 복사본으로 설정한다.
4. serialized.[[KeyingMaterialHandle]]을 value.[[KeyingMaterialHandle]]에 있는 핸들 직렬화로 설정한다.(개인 키 자료 자체가 아님)

주어진 직렬화 및 값에 대한 역직렬화 단계는 다음과 같다:

1. serialized.[[Expires]]를 포함하도록 value.expires 속성을 초기화한다.

2. serialized.[[Certificate]]를 value.[[Certificate]]의 복사본으로 설정한다.
3. value.[[Origin]]을 serialized.[[Origin]]의 복사본으로 설정한다.
4. value.[[KeyingMaterialHandle]]을 serialized.[[KeyingMaterialHandle]]을 역직렬화한 결과인 개인키 자료 핸들에 설정한다.

참고

이러한 방식으로 구조화된 복제를 지원하면 RTCCertificate 인스턴트를 저장소에 유지할 수 있다. 또한 postMessage()와 같은 API를 사용해서 인스턴스를 다른 출처에 전달할 수 있다. 그러나, 이 객체는 원래 객체를 생성한 원본이 아닌 다른 원본에서는 사용할 수 없다.

5) RTP 미디어 API

RTP 미디어 API는 웹 애플리케이션이 피어 투 피어 연결을 통해 MediaStreamTracks를 송수신할 수 있도록 한다. 트랙은 RTCPeerConnection에 추가될 때 신호를 발생시키며; 이 신호가 원격 피어로 전달될 때, 해당 트랙이 원격 측에 생성되게 한다.

참고

하나의 RTCPeerConnection에서 전송하고 다른 RTCPeerConnection에서 수신한 트랙 간에는 정확히 1:1의 대응성이 없다. 하나의 예를 들면, 전송된 트랙의 ID는 수신된 트랙의 ID에 매핑되지 않는다. 또한 replaceTrack은 수신자(receiver) 측에서 새 트랙을 만들지 않고 RTCRtpSender가 전송한 트랙을 변경한다. 해당 RTCRtpReceiver는 단일 트랙만 가지고 있으며, 이는 잠재적으로 여러 미디어 소스를 함께 연결한 것이다. addTransceiver와 replaceTrack 모두 동일한 트랙이 여러 번 전송되도록 할 수 있으며, 이는 각각 별도의 트랙을 가진 복수의 수신자로서 수신자 측에서 관측될 것이다. 따라서 필요한 경우 RTCRtpTransceiver의 mid를 사용하여 송신자와 수신자를 일치시키면서 한쪽의 RTCRtpSender와 다른 쪽의 RTCRtpReceiver의 트랙 간의 1:1 관계를 생각하는 것이 더 정확하다.

미디어를 보낼 때, 송신자는 SDP가 협상한 봉투를 포함한 다양한 요건을 충족하기 위해 미디어의 크기를 조정하거나 다시 샘플링해야 할 수 있다.

[JSEP] (섹션 3.6)의 규칙에 따라, 비디오는 SDP 제약 조건에 맞도록 축소될 수 있다(MAY). 입력 소스에서 발생하지 않은 가짜 데이터를 생성하기 위해 미디어를 확장해서는 절대로 안 되며(MUST NOT), 픽셀 수에 대한 제약을 충족하기 위해 필요한 경우를 제외하고 미디어를 잘라서는 절대로 안 되며(MUST NOT), 가로 세로 비율을 변경하면 절대로 안 된다(MUST NOT).

참고

WebRTC 작업 그룹은 이 상황의 보다 복잡한 처리를 위한 필요성과 일정에 대한 구현 피드백을 찾고 있다. GitHub 1283번째 이슈에서는 가능한 설계들이 논의되었다.

비디오가 재조정(rescaled)될 때, 예를 들어 비디오의 너비 또는 높이와 ScaleDefolutionDownBy 값의 특정 조합에 대한 경우, 너비 또는 높이의 결과가 정수가 아닌 상황이 발생할 수 있다. 이러한 상황에서 사용자 에이전트는 결과의 정수 부분을 사용해야 한다. 조정된 너비 또는 높이에서 정수 부분이 0일 경우 전송할 내용은 구현별로 다르다.

MediaStreamTracks의 실제 인코딩 및 전송은 RTCRtpSenders라는 객체를 통해 관리된다. 마찬가지로 MediaStreamTracks의 수신 및 디코딩은 RTCRtpReceivers라는 객체를 통해 관리된다. 각 RTCRtpSender는 최대 하나의 트랙과 연결되며, 수신될 각 트랙은 정확히 하나의 RTCRtpReceiver와 연결된다.

각 MediaStreamTrack의 인코딩 및 전송은 그 특성(비디오 트랙의 경우 너비, 높이 및 프레임 레이트, 오디오 트랙의 경우 볼륨, 샘플 크기, 샘플레이트 및 채널 카운트)이 원격에서 생성된 트랙에 의해 유지되도록 해야 한다(SHOULD). 이를 적용하지 않는 상황이 있을 수 있으며, 예를 들어 엔드포인트 또는 네트워크에 리소스 제약이 있을 수 있으며, 구현을 다르게 수행하도록 지시하는 RTCRtpSender 설정이 적용될 수 있다.

RTCPeerConnection 객체는 RTCRtpTransceiverser 집합을 포함하며, 일부 공유 상태를 가진 송신자와 수신자를 쌍으로 나타낸다. 이 집합은 RTCPeerConnection 객체가 생성될 때 빈 집합으로 초기화된다. RTCRtpSenders와 RTCRtpReceivers는 항상 RTCRtpTransceiver와 동시에 생성되며, 이 RTCRtpTransceiver는 생성이 유지될 동안 계속 부착될 것이다. RTCRtpTransceiverser는 애플리케이션이 addTrack() 메소드를 통해 RTCPeerConnection에 MediaStreamTrack을 부착할 때, 또는 addTransceiver 메소드를 사용할 때 명시적으로 생성된다. 또한 새 미디어 기술을 포함하는 원격 기술이 적용될 때 생성된다. 또한 원격

엔드포인트에 전송할 미디어가 있음을 나타내는 원격 기술이 적용되면 관련 MediaStreamTrack 및 RTCRtpReceiver가 트랙 이벤트를 통해 애플리케이션에 표면화된다.

RTCRtpTransceiver가 다른 엔드포인트와 미디어를 송수신하려면 두 엔드포인트 모두 동일한 미디어 기술과 연관된 RTCRtpTransceiver 객체를 갖도록 SDP를 통해 협상해야 한다.

오퍼(offer)를 생성할 때, 그 목적의 모든 트랜시버를 다루기에 충분한 미디어 기술이 생성될 것이다. 이 오퍼(offer)가 local description으로 설정되면, 관련되지 않은 모든 트랜시버는 오퍼(offer)의 미디어 설명과 연관된다.

오퍼(offer)가 remote description으로 설정되었다면, 트랜시버와 아직 관련되지 않은 미디어 기술은 새 트랜시버 또는 기존 트랜시버와 연관된다. 이 경우 addTrack() 메소드를 통해 생성된 연결되지 않은 트랜시버만 연결할 수 있다. 그러나 addTransceiver() 메소드을 통해 생성된 연결 해제된 트랜시버는 원격 오퍼(offer)에서 미디어 기술을 사용할 수 있더라도 연결되지 않는다. 대신에, 만약 addTrack()으로 생성된 트랜시버가 충분하지 않다면 새로운 트랜시버가 만들어지고 연결될 것이다. 이것은 속성 검사에서 관측할 수 없는 중요한 방법으로 addTrack() 또는 addTransceiver()으로 생성된 트랜시버를 분리한다.

Answer를 작성할 때 offer에 제시된 미디어 기술만 answer에 나열할 수 있다. 따라서 remote offer을 설정할 때 연결되지 않은 트랜시버는 local answer을 설정한 후에도 연결이 끊어진 상태로 남아 있다. 이는 후속 offer을 작성하거나, 다른 answer/offer 교환을 시작하거나, addTrack()에서 만든 트랜시버를 사용하는 경우, 초기 교환에서 충분한 미디어 기술이 제공되도록 함으로써 해결할 수 있다.

(1) RTCPeerConnection 인터페이스 확장

RTP 미디어 API는 아래에 설명된 RTCPeerConnection 인터페이스를 확장한다.

WebIDL

```
partial interface RTCPeerConnection {
 sequence<RTCRtpSender> getSenders();
 sequence<RTCRtpReceiver> getReceivers();
 sequence<RTCRtpTransceiver> getTransceivers();
 RTCRtpSender addTrack(MediaStreamTrack track, MediaStream... streams);
 undefined removeTrack(RTCRtpSender sender);
```

```
 RTCRtpTransceiver addTransceiver((MediaStreamTrack or DOMString)
          trackOrKind, optional RTCRtpTransceiverInit init = {});
 attribute EventHandler ontrack;
};
```

속성

ontrack, EventHandler 타입

이 이벤트 핸들러의 이벤트 유형은 트랙이다.

메소드

getSenders

이 RTCPeerConnection 객체에서 현재 연결되었고 중지되지 않은 RTCRtpTransceiver 객체에 속하는 RTP 송신자를 나타내는 RTCRtpSender 객체 시퀀스를 반환한다.

getSenders 메소드가 호출되면 사용자 에이전트는 반드시(MUST) CollectSenders 알고리즘을 실행한 결과를 반환해야 한다.

CollectSenders 알고리즘은 다음과 같이 정의한다:

1. 트랜시버는 CollectTransceivers 알고리즘을 실행한 결과물이 되도록 한다.
2. 송신자를 새로운 시퀀스로 설정한다.
3. 트랜시버들의 각 트랜시버에 대하여,
    1) transceiver.[[Stopped]]가 거짓이라면, transceiver.[[Sender]]를 송신자(sender)에 추가한다.
4. 송신자를 리턴한다.

getReceivers

이 RTCPeerConnection 객체에 현재 연결되었고 중지되지 않은 RTCRtpTransceiver 객체에 속하는 RTP 수신자를 나타내는 RTCRtpReceiver 객체의 시퀀스를 반환한다.

getReceivers 메소드가 호출되면 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야 한다:

1. 트랜시버를 CollectTransceivers 알고리즘을 실행한 결과물이 되도록 한다.

2. 수신자를 새로운 빈 시퀀스로 설정한다.
3. 트랜시버들의 각 트랜시버에 대하여,
   1) transceiver.[[Stopped]]가 거짓이라면, transceiver.[[Receiver]]를 수신자에 추가한다.
4. 수신자를 리턴한다.

getTransceivers

RTCPeerConnection 객체에 현재 부착된 RTP 트랜시버를 나타내는 RTCRtpTransceiver 객체의 시퀀스를 반환한다.

getTransceivers 메소드는 반드시(MUST) CollectTransceivers 알고리즘을 실행한 결과를 반환해야 한다.

CollectTransceivers 알고리즘을 다음과 같이 정의한다:

1. 트랜시버를 이 RTCPeerConnection 객체의 트랜시버 집합에 있는 모든 RTCRtpTransceiver 객체로 구성된 새로운 시퀀스가 되도록 한다.
2. 트랜시버를 리턴한다.

addTrack

RTCPeerConnection에 새 트랙을 추가하고 지정된 MediaStreams에 포함되었음을 표시한다. addTrack 메소드가 호출되면 사용자 에이전트는 다음 단계를 실행해야 한다.

1. 연결을 이 메소드가 호출된 RTCPeerConnection 개체로 설정한다.
2. 트랙은 메소드의 첫 번째 인자로 표시된 MediaStreamTrack 객체가 되도록 한다.
3. kind를 track.kind로 둔다.
4. 스트림을 메소드의 나머지 인자에서 생성된 MediaStream 객체의 목록으로 두거나 메소드가 단일 인자로 호출된 경우 빈 리스트로 둔다.
5. 만약 connection.[[isClosed]]가 true라면, InvalidStateError를 발생시킨다.
6. 송신자가 CollectSenders 알고리즘을 실행한 결과가 되도록 한다. 만약 트랙에 대한 RTCRtpSender가 이미 송신자에 있는 경우 InvalidAccessError를 발생(throw)시킨다.
7. 아래 단계에서는 기존 송신자를 재사용할 수 있는지 여부를 결정하는 방법을 설명한다. 수행 한다면, 향후의 통화(call)는, [JSEP](제5.2.2절 및 제5.3.2절)에서 정의한 대로, 해당 미디어 기술을 sendrecv이나 sendonly로 표시하고 송신자 스트림의 MSID를 추가하기

위해 createOffer와 createAnswer를 유발한다.

만약 송신자의 RTCRtpSender 객체가 다음 조건과 모두 일치하는 경우 송신자를 해당 객체로 두거나 그렇지 않으면 null로 둔다:

- 송신자의 트랙이 null이다.
- 송신자와 관련된 RTCRtpTransceiver의 트랜시버 kind는 kind와 일치한다.
- 송신자와 관련된 RTCRtpTransceiver의 [[Stopping]]슬롯은 false이다.
- 송신자는 송신에 사용한 적이 없다. 좀 더 정확히 말하면, 송신자와 연결된 RTCRtpTransceiver의 [[CurrentDirection] 슬롯은 "sendrecv" 또는 "sendonly"의 값을 가진 적이 없다.

8. 만약 송신자가 null이 아닌 경우 다음 단계를 실행하여 해당 송신자를 사용한다:
   1) sender.[[SendterTrack]]을 트랙으로 둔다.
   2) sender.[[AssociatedMediaStreamIds]]를 빈 집합으로 둔다.
   3) 스트림들의 각 스트림에서, 아직 steam.id가 존재하지 않는 경우, steam.id를 [[AssociatedMediaStreamIds]]에 추가한다.
   4) 트랜시버를 송신자와 관련된 RTCRtpTransceiver로 둔다.
   5) 만약 transceiver.[[Direction]]이 “recvonly”라면, transceiver.[[Direction]]을 “sendrecv”로 설정한다.
   6) 만약 transceiver.[[Direction]]이 “inactive””라면, transceiver.[[Direction]]을 “sendonly”로 설정한다.
9. 만약 송신자가 null이라면, 다음 단계를 수행한다.
   1) track, kind 그리고 stream이 있는 RTCRtpSender를 생성하고 송신자를 result로 둔다.
   2) kind와 함께 RTCRtpReceiver를 설정하고, 수신자를 result로 둔다.
   3) 송신자, 수신자 및 "sendrecv"의 RTCRtpTransceiverDirection 값을 사용하여 RTCRtpTransceiver를 생성하고 트랜시버를 result로 둔다.
   4) 연결의 트랜시버 집합에 트랜시버를 추가한다.
10. 트랙에는 응용 프로그램에 액세스할 수 없는 콘텐츠가 있을 수 있다. 이것은 CORS cross-origin을 유발하는 어떤 것 때문일 수 있다. 이러한 트랙은 addTrack() 메소드에 공급될 수 있으며, 이들을 위해 RTCRtpSender를 만들 수 있지만, 콘텐츠를 전송해서는

안 된다(MUST NOT). 음소거(오디오), 검은색 프레임(비디오) 또는 이와 동등하게 빈 콘텐츠가 트랙 콘텐츠 대신 전송된다.
참고로 이 속성은 시간이 지남에 따라 변경될 수 있다는 점에 유의해야 한다.

11. 연결을 위한 negotiation-needed 플래그를 업데이트한다.
12. 송신자를 리턴한다.

removeTrack

송신자으로부터 미디어 전송을 중지한다. RTCRtpSender는 여전히 getSenders에 나타날 것이다. 이렇게 하는 것은 [JSEP](섹션 5.2.2)에 정의된 대로 해당 트랜시버에 대한 미디어 설명을 "recvonly" 또는 "inactive"로 표시하기 위한 미래의 createOffer 호출이 생성될 것이다.

다른 피어가 이러한 방식으로 트랙 전송을 중지하면 트랙 이벤트에서 처음 공개된 원격 MediaStreams에서 트랙이 제거되고, MediaStreamTrack이 이미 음소거되지 않은 경우 트랙에서 음소거 이벤트가 발생하게 된다.

참고

removeTrack()과 같은 효과는 해당 트랜시버의 RTCRtpTransceiver.direction 속성을 설정하고 송신자에게 RTCRtpSender.replaceTrack(null)을 호출하여 얻을 수 있다. 한 가지 사소한 차이점은 replaceTrack()은 비동기식이고 removeTrack()은 동기식이라는 점이다.

removeTrack 메소드가 호출되면 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야 한다:

1. 송신자(sender)를 removeTrack의 인자로 둔다.
2. 연결이 메소드가 호출된 RTCPeerConnection 객체가 되도록 한다.
3. connection.[[IsClosed가 true라면, InvalidStateError를 발생(throw)시킨다.
4. 만약 송신자가 연결에 의하여 생성된 것이 아니라면, InvalidAccessError를 발생시킨다.
5. 송신자를 CollectSenders 알고리즘을 실행한 결과가 되도록 한다.
6. 만약 송신자가 송신자들(RTCSessionDescription 설정의 값이 “rollback” 때문에 트랜시버가 중지 또는 제거되었음을 나타낸다)에 없는 경우, 이 과정을 중단한다.
7. 만약 sender.[[SenderTrack]]이 null이라면, 이 과정을 중단한다.
8. sender.[[SenderTrack]]을 null로 둔다.

9. 트랜시버를 송신자에 해당하는 RTCRtpTransceiver 객체가 되도록 한다.
10. 만약 transceiver.[[Direction]]이 “sendrecv라면, transceiver.[[Direction]]을 “recvonly”로 둔다.
11. 만약 transceiver.[[Direction]]이 “sendonly”라면, transceiver.[[Direction]]을 “inactive”로 둔다.
12. 연결을 위한 negotiation-needed 플래그를 업데이트 한다.

addTransceiver

새로운 RTCRtpTransceiver를 만들어 트랜시버 집합에 추가한다.

트랜시버를 추가하면 해당 트랜시버에 대한 미디어 설명(description)을 추가하기 위해 createOffer에 대한 향후 호출(calls)을 발생시킨다. ([JSEP](섹션 5.2.2)에 정의)

mid의 초기 값은 null이다. 새 RTCSessionDescription을 세팅하면 null이 아닌 값으로 변경할 수 있고([JSEP](섹션 5.5 및 섹션 5.6)에 정의), RTCSessionDescription을 설정할 수 있다. sendEncodings 인자는 제안된(offered) 동시 캐스팅(simulcast) 인코딩의 수와 선택적으로 RID 및 인코딩 파라미터를 명시하는 데 사용될 수 있다.

이 메소드가 호출되면 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야 한다:

1. init을 두 번째 인자로 둔다.
2. streams를 init.streams로 둔다.
3. sendEncodings를 init.sendEncodings로 둔다.
4. direction을 init.direction으로 둔다.
5. 만약 첫 번째 인자가 문자열이라면, kind로 두고 다음 단계를 실행한다.
    1) 만약 kind가 적법한 MediaStreamTrack이 아니라면, TypeError를 발생(throw)시킨다.
    2) track을 null로 설정한다.
6. 만약 첫 번째 인자가 MediaStreamTrack이라면, 트랙으로 설정하고 kind를 track.kind로 둔다.
7. 만약 connection.[[IsClosed]]가 true라면, InvalidStateError를 발생시킨다.
8. sendEncodings를 다음 단계를 실행해서 확인한다.
    1) sendEncodings의 각 rid 값이 [MMUSIC-RID]의 섹션 10에 지정된 문법을 준수하는지

확인해야 한다. RID 중 하나가 이러한 요구 사항을 충족하지 못하면 TypeError를 발생시킨다.

2) 만약 sendEncodings의 RTCRtpEncodingParameters 딕셔너리에 rid 이외의 읽기 전용 파라미터가 포함되어 있는 경우 InvalidAccessError를 발생시킨다.

3) sendEncodings의 각 scaleDefolutionDownBy 값이 1.0보다 크거나 같은지 확인한다. ScaleDefolutionDownBy 값 중 하나가 이 요구 사항을 충족하지 못하면 RangeError를 발생시킨다.

4) maxN은 사용자 에이전트가 이러한 종류에 대해 지원할 수 있는 최대 동시 인코딩 수(최소 1개)가 되도록 한다. 사용될 코덱이 아직 알려지지 않았기 때문에 이것은 낙관적인 숫자여야 한다.

5) 만약 sendEncodings에 scaleDefolutionDownBy 특성이 정의된 인코딩이 포함된 경우 다른 인코딩의 정의되지 않은 scaleDefolutionDownBy를 1.0으로 설정한다.

6) 만약 sendEncoding에 저장된 RTCRtpEncodingParameter의 수가 maxN을 초과할 경우 마지막으로부터 sendEncoding의 길이가 maxN이 될 때까지 자른다.

7) sendEncodings의 scaleDefirmDownBy 속성이 여전히 정의되지 않은 경우, 각 인코딩의 scaleDefolutionDownBy를 2^(sendEncodings - 인코딩 인덱스 - 1)로 초기화한다. 이렇게 하면 마지막 인코딩에 스케일링이 적용되지 않는 해상도가 더 작아진다. 예를 들어 길이가 3인 경우 4:2:1과 같다.

8) 만약 현재 sendEncoding에 저장되어 있는 RTCRtpEncodingParameter의 수가 1인 경우, 임의의 rid 멤버를 유일한 입력에서 제거한다.

참고

sendEncodings에서 단일의 기본 RTCRtpEncodingParameter를 제공하면, 애플리케이션은 시뮬캐스트를 사용하지 않더라도 setParameter를 사용하여 후속적으로 인코딩 파라미터를 설정할 수 있다.

9) sendEncodings가 설정된 경우, createOffer를 위한 후속 호출은 [JSEP](섹션 5.2.2 및 섹션 5.2.1)에 정의된 대로 여러 개의 RTP 인코딩을 전송하도록 구성된다. setRemoteDescription이 [JSEP](섹션 3.7)에 정의된 대로 복수의 RTP 인코딩을 수신할 수 있는 해당 원격 기술로부터 호출될 때, RTCRTPSender는 복수의 RTP

인코딩을 보낼 수 있으며, 트랜시버의 sender.getParameters()를 통해 얻어진 파라미터는 협상된 인코딩을 반영한다.

10) RTCRtpReceiver를 kind와 함께 생성하여 수신자를 결과가 되도록 한다.

11) 송신자, 수신자 및 direction과 함께 RTCRtpTransceiver를 만들고 트랜시버를 결과로 만든다.

12) 연결의 트랜시버 집합에 트랜시버를 추가한다.

13) 연결을 위한 negotiation-needed 플래그를 업데이트한다.

14) 트랜시버를 리턴하다.

WebIDL

```
dictionary RTCRtpTransceiverInit {
 RTCRtpTransceiverDirection direction = "sendrecv";
 sequence<MediaStream> streams = [];
 sequence<RTCRtpEncodingParameters> sendEncodings = [];
};
```

RTCRtpTransceiverInit 딕셔너리 멤버

direction : RTCRtpTransceiverDirection 타입, 기본값 “sendrecv”

streams, RTCRtpTransceiver의 방향을 나타낸다.

streams : <MediaStream>시퀀스 타입

추가되는 RTCRtpReceiver에 해당하는 원격 PeerConnection의 트랙 이벤트가 발생할 때, 이러한 스트림이 이벤트에 포함될 스트림이다.

sendEncodings : <RTCRtpEncodingParameters>시퀀스 타입

미디어의 RTP 인코딩을 전송하기 위한 파라미터를 포함하는 시퀀스.

WebIDL

```
enum RTCRtpTransceiverDirection {
  "sendrecv",
  "sendonly",
  "recvonly",
  "inactive",
  "stopped"
};
```

열거형 RTCRtpTransceiverDirection 설명	
sendrecv	RTCRtpTransceiver의 RTCRtpReceiver는 RTP를 수신할 것을 offer하며, 원격 피어가 수락할 경우 RTP를 수신할 것이다.
sendonly	RTCRtpTransceiver의 RTCRtpSender 송신자는 RTP를 송신할 것을 offer하며, 원격 피어가 승인한다면 sender.getParameters().encodings[i].active가 참인 모든 i 값에 대하여 RTP를 송신할 것이다. RTCRtpTransceiver의 RTCRtpReceiver는 RTP를 수신할 것을 offer하지 않을 것이며, RTP를 수신하지 않을 것이다.
recvonly	RTCRtpTransceiver의 RTCRtpReceiver는 RTP를 수신할 것을 offer하며, 원격 피어가 수락할 경우 RTP를 수신할 것이다.
inactive	RTCRtpTransceiver의 RTCRtpSender 송신자는 RTP를 송신할 것을 offer하지 않을 것이며, RTP를 송신하지 않을 것이다. RTCRtpTransceiver의 RTCRtpReceiver는 RTP를 수신할 것을 offer하지 않을 것이며, RTP를 수신하지 않을 것이다.
stopped	RTCRtpTransceiver는 RTP를 송신하거나 수신하지 않는다. 그것은 offer에 대하여 포트 0번을 생성할 것이다. answer에서는, RTCRtpSender는 RTP 전송을 offer하지 않을 것이며, RTCRtpReceiver는 RTP를 받기 위한 offer을 하지 않을 것이다. 이것은 종료 상태이다.

① 원격 MediaStreamTracks 처리

애플리케이션은 트랜시버의 방향을 일시적으로 양방향을 끄기 위해 "inactive"으로 설정하거나, 수신측만 거부하도록 "sendonly"으로 설정하여 수신하는 미디어 기술을 거부할 수 있다. 재사용을 가능하게 하기 위하여 영구적으로 m-line을 거부하려면, 애플리케이션은 RTCRtpTransceiver.stop()을 호출한 후 마지막에서 협상을 개시해야 한다.

RTCRtpTransceiver 트랜시버, direction, msid, addList, removeList 및 trackEventInits가 지정된 원격 트랙을 처리하려면 다음 단계를 실행한다:

1. transceiver.[[Receiver]]와 원격 스트림을 설정한다. msid, addList 및 removeList.
2. 만약direction이 "sendrecv" 또는 "recvonly" 및 transceiver.[[FiredDirection]]가 "sendrecv"도 "recvonly"도 아닌 경우, 이전 단계에서 addList의 길이를 늘린 경우 trackEventInits를 사용하여 트랜시버와 함께 원격 트랙의 추가를 처리한다.
3. 만약 방향이 “sendonly” 또는 “inactive”인 경우, transceiver.[[Receptive]]를 false로 둔다.
4. 만약 direction이 "sendonly" 또는 "inactive" 그리고 transceiver.[[FiredDirection]]이 마찬가지로 "sendrecv" 또는 "recvonly"인 경우, 트랜시버 및 muteTracks를 사용하여 미디어 기술에 대한 원격 트랙의 제거를 처리한다.
5. transceiver.[[FiredDirection]]을 direction으로 설정

RTCRtpTransceiver 트랜시버 및 trackEventInits에 지정된 원격 트랙 추가를 처리하려면 다음 단계를 실행한다:

1. 수신자를 transceiver.[[Receiver]]로 둔다.
2. 트랙을 receiver.[[ReceiverTrack]]으로 둔다.
3. 스트림을 receiver.[[AssociatedRemoteMediaStreams]]로 둔다.
4. 리시버, 트랙, 스트림과 트랜시버를 멤버로 한 사전으로 새로운 RTCTrackEventInt를 만들어 trackEventInits에 추가한다.

RTCRtpTransceiver 트랜스시버 및 muteTracks를 사용하여 원격 트랙 제거를 처리하려면 다음 단계를 실행한다:

1. 수신자를 transceiver.[[Receiver]]로 둔다.
2. 트랙을 receiver.[[ReceiverTrack]]으로 둔다.
3. 만약 track.muted가 flase라면, 트랙을 muteTracks에 추가한다.

RTCRtpReceiver 수신자, msid, addList 및 removeList가 지정된 관련 원격 스트림을 설정하려면 다음 단계를 실행한다:

1. 연결(Connection)을 수신자와 연결된 RTCPeerConnection 객체로 둔다.
2. msid들의 각 MSID에 대해 MediaStream 객체가 이전에 이 연결에 대해 해당 ID를 사용하여 생성되지 않은 경우 해당 ID를 사용하여 MediaStream 객체를 생성하십시오.
3. 스트림을 msid에 해당하는 ID와 연결하기 위해 생성된 MediaStream 객체의 리스트로 설정하십시오.
4. 수신자를 receiver.[[Receiver]]로 둔다.
5. 스트림에 없는 receiver.[[AssociatedRemoteMediaStreams]]에 대하여 스트림과 트랙을 쌍으로 removeList에 추가한다.
6. receiver.[[AssociatedRemoteMediaStreams]]에 없는 스트림들의 각각의 스트림에 대하여, 스트림과 트랙을 쌍으로 addList에 추가한다.
7. receiver.[[AssociatedRemoteMediaStreams]]를 streams에 설정한다.

(2) RTCRtpSender 인터페이스

RTCRtpSender 인터페이스는 응용 프로그램이 주어진 MediaStreamTrack이 인코딩되고 원격 피어로 전송되는 방법을 제어할 수 있게 해준다. RTCRtpSender 객체에서 setParameter를 호출하면 인코딩이 적절하게 변경된다.

MediaStreamTrack, 트랙, 문자열, kind, MediaStream 객체 리스트, 스트림 그리고 선택적으로 RTCRtpEncodingParameter 객체, sendEncodings를 사용하여 RTCRtpSender를 생성하려면 다음 단계를 실행한다:

1. sender를 새로운 RTCRtpSender 객체로 둔다.
2. sender에게 내부 슬롯이 초기화된 [[SenderTrack]]을 갖게 한다.
3. sender에게 내부 슬롯이 null로 초기화된 [[SenderTransport]]를 갖게 한다.
4. sender에게 내부 슬롯이 null로 초기화된 [[LastStableStateSenderTransport]]를 갖게 한다.
5. sender에게 내부 슬롯이 null로 초기화된 [[Dtmf]]를 갖게 한다.
6. 만약 kind가 “audio”라면 RTCDTMFSender dtmf를 생성하고 [[Dtmf]] 내부 슬롯을 dtmf로 설정한다.

7. sender에게 [[AssociatedMediaStreamIds]] 내부 슬롯을 갖게 한다. 이 슬롯은 이 sender가 연결할 MediaStream 객체의 ID 목록을 나타낸다. [[AssociatedMediaStreamIds] 슬롯은 sender가 [JSEP](섹션 5.2.1)에 설명된 대로 SDP에 표시될 때 사용된다.
8. sender.[[AssociatedMediaStreamIds]]를 빈 집합으로 설정한다.
9. 스트림들에 있는 각 스트림마다, stream.id가 이미 존재하지 않는다면 [[AssociatedMediaStreamIds]]에 추가한다.
10. sender가 RTCRtpEncodingParameters 딕셔너리의 리스트를 의미하는 [[SendEncodings]] 내부 슬롯을 갖게 한다.
11. 만약 SendEncodings가 이 알고리즘의 입력으로 제공되고 비어 있지 않은 경우 [[SendEncodings]] 슬롯을 sendEncodings로 설정한다. 그렇지 않으면 활성 집합이 true인 단일 RTCRtpEncodingParameter를 포함하는 리스트로 설정한다.
12. sender가 빈 리스트로 초기화된 RTCRtpCodecParameters 딕셔너리의 리스트를 의미하는 [[SendCodes]] 내부 슬롯을 갖게 한다.
13. sender가 getParameters와 setParameters 트랜잭션을 맞추기 위한 [[LastReturnedParameters]] 내부 슬롯을 갖게 한다.)
14. sender를 리턴한다.

WebIDL

```
[Exposed=Window]
interface RTCRtpSender {
 readonly attribute MediaStreamTrack? track;
 readonly attribute RTCDtlsTransport? transport;
 static RTCRtpCapabilities? getCapabilities(DOMString kind);
 Promise<undefined> setParameters(RTCRtpSendParameters parameters);
 RTCRtpSendParameters getParameters();
 Promise<undefined> replaceTrack(MediaStreamTrack? withTrack);
 undefined setStreams(MediaStream... streams);
 Promise<RTCStatsReport> getStats();
};
```

속성

track : MediaStreamTrack 타입, 읽기전용, null가능,

track 속성은 이 RTCRtpSender 객체와 연결된 트랙이다. 트랙이 종료되거나 트랙의 출력이 비활성화된 경우, 즉 트랙이 비활성화되거나 음소거된 경우, RTCRtpSender는 검은색 프레임(비디오)을 반드시 전송해야 하며, 오디오를 전송해서는 반드시 안 된다. 비디오의 경우 RTCRtpSender는 초당 1개의 검은색 프레임을 전송해야 한다. 트랙이 null이면 RTCRtpSender가 전송하지 않는다. 가져올 때의 속성은 [SenderTrack] 슬롯 값을 반드시 반환해야 한다.

transport : RTCDtlsTransport 타입, 읽기전용, null가능

transport 속성은 트랙에서 미디어가 RTP 패킷의 형태로 전송되는 전송이다. RTCDtls Transport 객체를 구성하기 전에 transport 속성은 null이 될 것이다. 번들이 사용될 때, 복수의 RTCRtpSender 객체는 하나의 transport를 공유하며, 모두 동일한 transport를 통해 RTP와 RTCP를 전송한다.

가져올 때의 속성은 [[SenderTransport]] 슬롯 값을 반드시 반환해야 한다.

메소드

getCapabilities : 정적 메소드

getCapability() 메소드는 주어진 종류의 미디어를 전송하기 위한 시스템의 기능에 대한 가장 낙관적인 관점을 반환한다. 리소스, 포트 또는 기타 상태를 예약하지 않지만 지원되는 코덱을 포함하여 브라우저의 기능 유형을 탐색할 수 있는 방법을 제공하기 위한 것이다. 사용자 에이전트는 반드시 "audio"와 "video"의 종류 값을 지원해야 한다. 시스템에 해당 인자의 값에 해당하는 기능이 없는 경우 getCapability는 null을 반환한다.

이러한 기능은 일반적으로 장치에 영구적인 cross-origin 정보를 제공하므로 애플리케이션의 핑거프린트 측면을 향상시킨다. 프라이버시에 민감한 맥락에서 브라우저는 기능의 공통 부분 집합만 보고하는 것과 같은 완화를 고려할 수 있다.

setParameters

setParameters 메소드는 트랙이 인코딩되고 원격 피어로 전송되는 방법을 업데이트한다. setParameters 메소드가 호출되면 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야 한다.

1. 파라미터를 메소드의 첫 번째 인자로 둔다.

2. setParameters가 호출되면 sender를 RTCRtpSender 객체로 둔다.
3. 트랜시버를 sender과 관련된 RTCRtpTransceiver 객체로 둔다. [sender는 transceiver.[[Sender]]이다.
4. 만약 transceiver.[[Stopped]]가 true라면, 새롭게 생성된 InvalidStateError와 함께 reject된 promise를 리턴한다.
5. 만약 sender.[[LastReturnedParameters]]가 null이라면, 새롭게 생성된 InvalidStateError와 함께 reject된 promise를 리턴한다.
6. 다음 과정을 실행하여 파라미터를 검증한다.:
   1) encodings를 parameters.encodings로 둔다.
   2) codecs를 parameters.codec로 둔다.
   3) sender.[[SendEncodings]]에 저장된 RTCRtpEncodingParameters의 개수로 N을 설정한다.
   4) 만약 다음과 같은 조건이 성립된다면, 새롭게 생성된 InvalidModificationError와 함께 reject된 promise를 리턴한다:
      (1) encodings.length와 N이 다르다.
      (2) encodings가 다시 조정된다.
      (3) parameters에 있는 어느 파라미터라도 읽기전용 파라미터이다 (예를 들어 RID) 그리고 sender에 대응되는 파라미터와 다른 값을 가지고 있다. transactionId에도 적용되는 것을 참고하라.)
   5) encodings의 scaleResolutionDownBy의 값이 1.0과 같거나 크다는것을 검증하라. scaleResolutionDownBy의 값이 하나라도 조건에 맞지 않다면, 새롭게 생성된 RangeError와 함께 reject된 promise를 리턴한다.
7. p를 새로운 promise로 둔다.
8. 동시에, sender.[[SenderTrack]]을 전송하기 위해 사용하는 parameters를 사용하기 위해 미디어 스택을 구성한다.
   1) 만약 미디어 스택이 parameters를 통해 성공적으로 설정되었다면, 다음 과정을 실행하기 위한 작업을 대기열에 추가한다.)
      (1) sender.[[LastReturnedParameters]]를 null로 설정한다.
      (2) sender.[[SendEncodings]]를 parameters.encodings로 설정한다.

(3) undefined로 p를 해결한다.

2) 만약 미디어 스택을 구성하는 동안 오류가 발생하면 다음 단계를 실행하도록 작업을 대기열에 넣어야 한다:

(1) 만약 하드웨어 리소스를 사용할 수 없어 오류가 발생한 경우, errorDetail이 "hardware-encoder-not-available"로 설정된 새로 생성된 RTCError로 p를 reject 후, 단계를 중단한다.

(2) 만약 하드웨어 인코더가 파라미터를 지원하지 않아 오류가 발생한 경우, errorDetail이 "hardware-encoder-error"로 설정된 새로 생성된 RTCError로 p를 reject, 단계를 중단한다.

(3) 다른 모든 오류에 대해, 새롭게 생성된 OperationError로 p를 거부(reject)한다.

9. Return p. (p를 리턴.)

setParameter는 SDP 재협상을 유발하지 않으며, offer/answer으로 협상한 envelope 내에서 미디어 스택이 송신하거나 수신하는 내용을 변경하는 데만 사용할 수 있다.

RTCRtpSendParameters 딕셔너리의 속성은 이것을 활성화하지 않도록 설계되어 있으므로 변경할 수 없는 cname과 같은 속성은 읽기 전용이다. 비트 전송률과 같은 다른 것들은 사용자 에이전트가 최대 비트 전송률을 초과하지 않도록 하는 동시에 SDP와 같은 다른 곳에서 지정된 비트 전송률에 대한 제약을 만족시키도록 해야 하는 maxBitrate와 같은 한계를 사용하여 제어된다.

getParameters

getParameters() 메소드는 트랙이 인코딩되고 원격 RTCRtpReceiver로 전송되는 방법에 대한 RTCRtpSender 객체의 현재 파라미터를 반환한다.

getParameters가 호출되면, 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야 한다.:

1. sender를 수신자가 호출된 RTCRtpSender 객체로 설정하십시오.
2. 만약 sender.[[LastReturnedParameters]]가 null이 아니라면, sender.[[LastReturned Parameters]]를 리턴하고 단계를 중단한다.
3. result를 다음 과정을 통하여 구성된 새로운 RTCRtpSendParameters 딕셔너리로 둔다.:
   - transactionId를 새로운 고유 식별자로 설정한다.
   - encodings를 [[SendEncodings]]의 내부 슬롯의 값으로 설정한다.
   - headerExtensions 시퀀스는 전송을 위해 협상된 헤더 확장자를 기반으로 작성된다.

- codecs를 [[SendCodes]] 내부 슬롯의 값으로 설정한다.
- rtcp.cname은 관련 RTCPeerConnection의 CNAME으로 설정된다. rtcp.reducedSize가 전송을 위해 협상된 경우 true로 설정되고 그렇지 않은 경우 거짓로 설정된다.

4. sender.[[LastReturnedParameters]]를 result로 설정한다.
5. sender.[[LastReturnedParameters]]를 null로 설정하는 작업을 대기열에 추가한다.
6. 결과를 리턴한다.

getParameter를 setParameter와 함께 사용하여 다음과 같은 방법으로 파라미터를 변경할 수 있다:

Example 2

```
async function updateParameters() {
  try {
    const params = sender.getParameters();
    // ... make changes to parameters
    params.encodings[0].active = false;
    await sender.setParameters(params);
  } catch (err) {
    console.error(err);
  }
}
```

setParameter에 대한 호출이 완료된 후 getParameter에 대한 후속 호출은 수정된 파라미터 집합을 반환한다.

replaceTrack

RTCRtpSender의 현재 트랙을 재협상 없이 제공된 트랙(또는 null 트랙 포함)으로 대체하려는 시도를 한다.

replaceTrack 메소드가 호출되면 사용자 에이전트는 반드시 다음 단계를 실행해야 한다:

1. sender를 replaceTrack이 호출되는 RTCRtpSender 객체로 설정한다.
2. transceiver를 sender와 연관된 RTCRtpTransceiver 객체가 되도록 한다.

3. connection을 sender와 연관된 RTCPeerConnection 객체가 되도록 한다.
4. withTrack을 이 메소드의 인자가 되도록 한다.
5. 만약 withTrack이 null이 아님과 동시에 withTrack.kind가 transceiver의 트랜시버 종류와 다르다면, 새롭게 생성된 TypeError와 함께 reject된 promise를 리턴한다.
6. connection의 운영 체인에 다음 단계를 연결한 결과를 반환한다.:
   1) 만약 transceiver.[[Stopped]]가 true 라면, 새롭게 생성된 InvalidStateError와 함께 reject된 promise를 리턴한다.
   2) p를 새로운 promise로 둔다.
   3) transceiver.[[CurrentDirection]]이 "sendrecv" 또는 "sendonly"라면 sending을 true로 둔다. 그렇지 않으면 false로 둔다.
   4) 다음 단계를 병렬로 처리한다.:
      (1) 만약 sending이 참이고 withTrack이 null이라면, 송신자에게 송신을 중지시킨다.
      (2) 만약 sending이 true이고 withTrack이 null이 아니라면, withTrack이 송신자의 이미 협상된 봉투를 위반하지 않고 송신자에 의해 즉시 전송될 수 있는지 여부를 결정하고, 그렇지 않으면 새로 생성된 InvalidModificationError와 함께 rejected 된 promise를 리턴한다.
      (3) sending이 true이고 withTrack이 null이 아니라면, 송신자의 기존 트랙이 아닌 withTrack으로 매끄럽게 전환한다.
      (4) 다음 단계를 실행하는 작업을 대기열에 추가한다.:
         1〉 만약 connection.[[IsClosed]]가 true라면, 단계를 중단한다.
         2〉 sender.[[SenderTrack을 withTrack으로 설정한다.
         3〉 p를 undefined로 해결한다.
      (5) p를 리턴한다.

참고

비율 및/또는 프레임률을 변경하는 경우 협상이 필요하지 않을 수 있다. 협상이 필요할 수 있는 경우는 다음 포함한다.

1. [RFC6236]에 설명된 대로 협상된 이미지 범위 밖의 값으로 해상도 변경.
2. 프레임률을 코덱의 차단률를 초과하는 값으로 변경.
3. 원시 포맷과 사전 인코딩된 포맷의 비디오 트랙이 다른 경우
4. 오디오 트랙이 다른 채널 수를 가진 경우
5. 또한 인코딩하는 소스(일반적으로 하드웨어 인코더)는 협상된 코덱을 생산할 수 없을 수 있다. 마찬가지로 소프트웨어 소스는 인코딩 소스에 대해 협상된 코덱을 구현하지 못할 수 있다.

setStreams

이 송신자의 트랙과 연결할 MediaStream을 설정

setStreams 메소드가 호출되면 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야 한다.

1. sender를 메소드가 호출된 RTCRtpSender 객체로 설정한다.
2. connection을 이 메소드가 호출된 RTCPeerConnection 객체로 설정한다.
3. 만약 connection.[[IsClosed]]가 true라면, InvalidStateError를 발생시킨다.
4. 스트림을 메소드의 인수로 구성된 MediaStream 객체 리스트로 두거나, 메소드가 인자 없이 호출된 경우 빈 리스트로 둔다.
5. sender.[[AssociatedMediaStreamsIds를 빈 집합으로 설정한다.
6. 스트림들의 각 스트림에서, stream.id가 설정되지 않았다면 [[AssociatedMediaStreamIds]]로 설정한다.
7. connection을 위하여 negotiation-needed 플래그를 업데이트 한다.

getStats

송신자에 대한 통계만 수집하고 결과를 비동기식으로 보고한다.

getStats() 메소드가 호출되면 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야 한다:

1. 선택자(selector)를 메소드가 호출된 RTCRtpSender 객체로 둔다.
2. p를 새로운 promise로 두고, 병렬로 다음 단계를 실행한다.
   1) 통계 선택 알고리즘에 따라 선택자(selector)로 표시된 통계를 수집한다.
   2) 수집된 통계를 포함하는 결과인 RTCStatsReport 객체로 p를 결정한다.
3. p를 리턴한다.

① RTCRtpParameters 사전

WebIDL

```
dictionary RTCRtpParameters {
 required sequence<RTCRtpHeaderExtensionParameters> headerExtensions;
 required RTCRtcpParameters rtcp;
 required sequence<RTCRtpCodecParameters> codecs;
};
```

RTCRtpParameters 딕셔너리 멤버

headerExtensions : <RTCRtpHeaderExtensionParameters>시퀀스 타입, 요구됨

RTP 헤더 확장에 대한 파라미터를 포함하는 시퀀스. 읽기 전용 파라미터.

rtcp : RTCRtcpParameters 타입의 rtcp, 요구됨

RTCP에 사용하는 파라미터. 읽기 전용 파라미터.

codecs : RTCRtpCodecParameters 시퀀스 타입, 요구됨

RTCRtpSender가 선택할 미디어 코덱과 RTX, RED 및 FEC 메커니즘의 항목을 포함하는 시퀀스. RTX를 통한 재전송이 가능한 각 미디어 코덱에 대응하여, mimeType에 audio/rtx 또는 video/rtx를 통한 재전송을 나타내는 속성과 sdpFmtpLine 속성("apt" 및 "rtx-time" 파라미터 제공)이 codecs에 엔트리가 있을 것이다. 읽기 전용 파라미터.

② RTCRtpSendParameters 딕셔너리

WebIDL

```
dictionary RTCRtpSendParameters : RTCRtpParameters {
required DOMString transactionId;
required sequence<RTCRtpEncodingParameters> encodings;
};
```

RTCRtpSendParameters 딕셔너리 멤버

transactionId : DOMString타입, 요구됨

마지막으로 적용된 매개 변수 집합에 대한 고유 식별자. setParameter를 이전 get Parameter에 기반하여서만 호출할 수 있고, 간섭적인 변경이 없음을 보장한다. 읽기 전용 파라미터.

encodings : RTCRtpEncodingParameters>시퀀스 타입의 encodings, 요구됨

미디어의 RTP 인코딩에 대한 파라미터를 포함하는 시퀀스.

③ RTCRtpReceiveParameters 딕셔너리

WebIDL

```
dictionary RTCRtpReceiveParameters : RTCRtpParameters {
};
```

④ RTCRtpCodingParameters 딕셔너리

WebIDL

```
dictionary RTCRtpCodingParameters {
 DOMString rid;
};
```

RTCRtpCodingParameters 딕셔너리 멤버

rid : DOMString 타입

만약 설정되어있다면, 이 RTP 인코딩은 [JSEP](섹션 5.2.1)에서 정의한 RID 헤더 확장자와 함께 전송된다. RID는 setParameters를 통해 수정할 수 없다. 송신측 addTransceiver에서만 설정하거나 수정할 수 있다. 읽기 전용 파라미터.

⑤ RTCRtpDecodingParameters 딕셔너리

WebIDL

```
dictionary RTCRtpDecodingParameters : RTCRtpCodingParameters {};
```

⑥ RTCRtpEncodingParameters 딕셔너리

WebIDL

```
dictionary RTCRtpEncodingParameters : RTCRtpCodingParameters {
 boolean active = true;
 unsigned long maxBitrate;
 double scaleResolutionDownBy;
};
```

RTCRtpEncodingParameters 딕셔너리 멤버

active : boolean 타입의 active, 기본값 true

이 인코딩이 현재 전송 중임을 나타낸다. false로 설정하면 이 인코딩이 더 이상 전송되지 않는다. true로 설정하면 이 인코딩이 전송된다. 값을 false로 설정해도 SSRC가 제거되지 않으므로 RTCP BYE는 전송되지 않는다.

maxBitrate : unsigned long 타입

존재하는 경우, 이 인코딩을 전송하는 데 사용할 수 있는 최대 비트 전송률을 나타낸다. 사용자 에이전트는 maxBitrate 값을 초과하지 않는 한 인코딩 간에 대역폭을 자유롭게 할당할 수 있다.

또한 인코딩은 여기에 지정된 최대치 이하의 다른 한계(예를 들어 전송당 또는 세션당 대역폭 한계치)에 의해 더욱 제한될 수 있다.maxBitrate는 [RFC3890] 섹션 6.2.2에 정의된 IP 또는 TCP 또는 UDP와 같은 다른 전송 계층을 계산하지 않고 필요한 최대 대역폭을 의미하는 전송 독립 애플리케이션별 최대값(TIAS) 대역폭과 동일한 방법으로 계산된다. maxBitrate의 단위는 초당 비트 수이다.

참고

비트 전송률이 달성되는 방법은 미디어와 인코딩에 따라 달라진다. 비디오의 경우 프레임은 항상 가능한 빨리 전송되지만, 비트 전송률이 충분히 낮을 때까지 프레임은 삭제될 수 있다. 따라서 0의 비트레이트라도 하나의 프레임을 보낼 수 있게 된다. 오디오의 경우 비트 전송률이 선택한 인코딩을 전송하기에 충분한 대역폭을 허용하지 않을 경우 재생을 중지해야 할 수 있다.

scaleResolutionDownBy : double 타입

이 멤버은 송신자의 kind가 "video"인 경우에만 존재한다. 비디오의 해상도는 전송하기 전에 주어진 값에 의해 각 비율에서 축소될 것이다. 예를 들어 값이 2.0일 경우 각 차원마다 2배수씩 비디오의 크기를 줄여 1/4 크기의 비디오를 전송하게 된다. 값이 1.0이면 비디오는 영향을 받지 않는다. 값은 1.0보다 크거나 같아야 한다. 기본적으로 스케일링은 예를 들어 4:2:1과 같이 작은 해상도에서 높은 해상도 순서로 레이어 수의 2배에 따라 적용된다. 레이어가 하나만 있는 경우 송신자는 기본적으로 스케일링을 적용하지 않는다(예: ScaleDefolutionDownBy이 1.0 될 것이다).

⑦ RTCRtcpParameters 딕셔너리

WebIDL

```
dictionary RTCRtcpParameters {
 DOMString cname;
 boolean reducedSize;
};
```

RTCRtcpParameters 딕셔너리 멤버

cname : DOMString 타입

RTCP에서 사용하는 표준 이름(CNAME) (예: SDES 메시지). 읽기 전용 파라미터

reducedSize : boolean 타입

줄어든 크기의 RTCP [RFC5506]이(가) 구성되었는지(참일 경우) 또는 [RFC3550](거짓일 경우)에 명시된 복합 RTCP가 구성되었는지 여부. 읽기 전용 파라미터.

⑧ RTCRtpHeaderExtensionParameters 딕셔너리

WebIDL

```
dictionary RTCRtpHeaderExtensionParameters {
 required DOMString uri;
 required unsigned short id;
 boolean encrypted = false;
};
```

RTCRtpHeaderExtensionParameters 딕셔너리 멤버

uri : DOMString 타입, 요구됨

[RFC5285]에 정의된 RTP 헤더 확장의 URI. 읽기 전용 파라미터.

id : unsigned short 타입, 요구됨

헤더 확장을 식별하기 위해 RTP 패킷에 넣은 값. 읽기 전용 파라미터.

encrypted : Boolean 타입

헤더 확장명의 암호화 여부. 읽기 전용 파라미터.

참고

RTCRtpHeaderExtensionParameters 딕셔너리을 통해 애플리케이션은 헤더 확장이 RTCRtpSender 또는 RTCRtpReceiver 내에서 사용하도록 구성되었는지 여부를 결정할 수 있다. RTCRtpTransceiver 트랜시버의 경우, 애플리케이션은 SDP를 구문 분석할 필요 없이 다음과 같이 헤더 확장의 "direction" 매개변수([RFC5285]의 섹션 5에서 정의됨)를 결정할 수 있다.

1. sendonly: transceiver.sender.getParameters().headerExtensions만 포함된 헤더 확장.
2. recvonly: transceiver.receiver.getParameters().headerExtensions만 포함된 헤더 확장.
3. sendrecv: transceiver.sender.getParameters().headerExtensions와 transceiver.receiver.getParameters().headerExtensions를 둘 다 포함한 헤더 확장.
4. inactive: transceiver.sender.getParameters().headerExtensions 또는 transceiver.receiver.getParameters().headerExtensions를 포함하지 않은 헤더 확장

⑨ RTCRtpCodecParameters 딕셔너리

WebIDL

```
dictionary RTCRtpCodecParameters {
 required octet payloadType;
 required DOMString mimeType;
 required unsigned long clockRate;
 unsigned short channels;
 DOMString sdpFmtpLine;
};
```

RTCRtpCodecParameters 딕셔너리 멤버

payloadType : octet 타입, 요구됨

이 코덱을 식별하기 위해 사용하는 RTP 페이로드 유형. 읽기 전용 파라미터.

mimeType : DOMString 타입, 요구됨

코덱 MIME 미디어 유형/하위 유형. 유효한 미디어 유형과 하위 유형은 [IANA-RTP-2]에 열거되어 있다. 읽기 전용 파라미터.

clockRate : unsigned long 타입, 요구됨

코덱 클럭 속도는 헤르츠로 표현됨. 읽기 전용 파라미터.

channels : unsigned short 타입

존재하는 경우 채널 수(모노=1, 스테레오=2)를 나타낸다. 읽기 전용 파라미터.

sdpFmtpLine : DOMString 타입

[JSEP](섹션 5.8) 코덱에 정의된 대로 코덱에 해당하는 SDP의 a=fmtp 라인에서 "특정 파라미터 포맷" 필드. RTCRtpSender의 경우 이러한 파라미터는 원격 기술에서, RTCRtp Receiver의 경우 로컬 기술에서 나온다. 읽기 전용 파라미터.

⑩ RTCRtpCapabilities 디셔너리

WebIDL

```
dictionary RTCRtpCapabilities {
 required sequence<RTCRtpCodecCapability> codecs;
 required sequence<RTCRtpHeaderExtensionCapability> headerExtensions;
};
```

RTCRtpCapabilities 딕셔너리 멤버

codecs : RTCRtpCodecCapability>시퀀스 타입, 요구됨

지원되는 미디어 코덱과 RTX, RED 및 FEC 메커니즘의 항목. RTX를 통한 재전송을 위한 codecs에는 단 하나의 엔트리만 존재하고, sdpFmtpLine은 존재하지 않는다.

headerExtensions : RTCRtpHeaderExtensionCapability시퀀스 타입, 요구됨

RTP 헤더 확장 지원

⑪ RTCRtpCodecCapability 딕셔너리

WebIDL

```
dictionary RTCRtpCodecCapability {
 required DOMString mimeType;
 required unsigned long clockRate;
 unsigned short channels;
 DOMString sdpFmtpLine;
};
```

RTCRtpCodecCapability 딕셔너리 멤버

RTCRtpCodecCapability 딕셔너리는 코덱 기능에 대한 정보를 제공한다. 생성된 SDP offer에서 구별되는 페이로드 유형을 활용할 수 있는 기능 조합만 제공된다. 예를 들면 다음과 같다.

1. 지원되는 두 패킷화 모드 값 각각에 하나씩, H.264/AVC 코덱 2개.
2. 클럭 속도가 다른 두 CN 코덱.

mimeType : DOMString 타입, 요구됨

코덱 MIME 미디어 유형/하위 유형. 유효한 미디어 유형과 하위 유형은 [IANA-RTP-2]에 열거되어 있다.

clockRate : unsigned long 타입, 요구됨

코덱 클럭 속도는 헤르츠로 표현됨.

channels : unsigned short 타입

존재하는 경우, 최대 채널 수(모노=1, 스테레오=2)를 나타낸다.

sdpFmtpLine : DOMString 타입

코덱에 해당하는 SDP의 a=fmtp 라인(있는 경우)에서 "특정 파라미터 포맷" 필드.

⑫ RTCRtpHeaderExtensionCapability 딕셔너리

WebIDL

```
dictionary RTCRtpHeaderExtensionCapability {
 DOMString uri;
};
```

RTCRtpHeaderExtensionCapability 딕셔너리 멤버

uri : DOMString 타입

[RFC5285]에 정의된 RTP 헤더 확장자의 URI.

(3) RTCRtpReceiver 인터페이스

RTCRtpReceiver 인터페이스는 응용 프로그램이 MediaStreamTrack의 수신 상태를 검사할 수 있도록 허용한다.

string, kind를 사용하여 RTCRtpReceiver를 만들려면 다음 단계를 실행한다:

1. receiver를 새로운 RTCRtpReceiver 객체로 설정한다.
2. 트랙을 새로운 MediaStreamTrack 객체 [GETUSERMEDIA]로 설정한다. 트랙의 소스는 receiver가 제공하는 원격 소스가 된다. track.id은 사용자 에이전트에 의해 생성되며 원격의 트랙 ID에는 매핑되지 않는다는 점에 유의한다.
3. track.kind를 kind로 초기화한다.
4. track.label을 "remote" 문자열을 kind로 병합한 결과로 초기화한다.
5. track.readyState를 live로 초기화한다.
6. track.muted를 true로 둔다. 음소거된 MediaStreamTrack이 미디어 데이터를 수신하는 경우 음소거된 속성이 어떻게 반영되는지 여부는 MediaStreamTrack 섹션을 참조한다.
7. receiver에게 [[ReceiverTrack]] 내부 슬롯을 track으로 초기화한 값을 갖도록 한다.
8. receiver에게 [[ReceiverTransport]] 내부 슬롯을 null로 초기화한 값을 갖도록 한다.
9. receiver에게 [[LastStableStateReceiverTransport]] 내부 슬롯을 null로 초기화한 값을 갖도록 한다.

10. receiver에게 [[AssociatedRemoteMediaStreams]] 내부 슬롯을 갖게 하고, 이 receiver의 MediaStreamTrack 객체가 연결된 MediaStream 객체 리스트를 나타내고 빈 리스트로 초기화한다.
11. receiver에게 [[LastStableStateAssociatedRemoteMediaStreams]] 내부 슬롯을 갖게 하고 빈 리스트로 초기화한다.
12. receiver에게 RTCRtpCodecParameters 딕셔너리의 리스트를 의미하는 [[Receive Codecs]] 내부 슬롯을 갖게 하고, 빈 리스트로 초기화한다.
13. receiver에게 [[LastStableStateReceiveCodecs]] 내부 슬롯을 갖게하고 빈 리스트로 초기화한다.
14. receiver를 리턴한다.

WebIDL

```
[Exposed=Window]
interface RTCRtpReceiver {
 readonly attribute MediaStreamTrack track;
 readonly attribute RTCDtlsTransport? transport;
 static RTCRtpCapabilities? getCapabilities(DOMString kind);
 RTCRtpReceiveParameters getParameters();
 sequence<RTCRtpContributingSource> getContributingSources();
 sequence<RTCRtpSynchronizationSource> getSynchronizationSources();
 Promise<RTCStatsReport> getStats();
};
```

속성

track : MediaStreamTrack 타입, 읽기 전용

track 속성은 이 RTCRtpReceiver 객체 receiver와 연결된 트랙이다.

참고로 클론은 영향을 받지 않지만 track.stop()이 최종이라는 것을 참고해야한다.

receiver.track.stop()은 receiver를 암시적으로 중지하지 않으므로, receiver는 리포트는 계속적으로 전송된다. 가져올 때, 속성은 반드시 [[ReceiverTrack]] 슬롯의 값을 리턴해야 한다.

transport : RTCDtlsTransport 타입, 읽기 전용, null 가능

transport 속성은 수신자의 트랙을 위한 미디어가 RTP 패킷의 형태로 수신되는 전송이다.

RTCDtlsTransport 객체를 구성하기 전에 transport 속성은 null이 될 것이다. 번들을 사용할 때, 복수의 RTCRtpReceiver 객체는 하나의 transport를 공유하며, 모두 동일한 전송을 통해 RTP와 RTCP를 수신한다.

가져올 때, 속성은 반드시(MUST) [ReceiverTransport] 슬롯 값을 리턴해야 한다.

메소드

getCapabilities : 정적 메소드

getCapability() 메소드는 주어진 종류의 미디어를 수신하기 위한 시스템의 기능에 대한 가장 낙관적인 관점을 반환한다. 리소스, 포트 또는 기타 상태를 예약하지 않지만 지원되는 코덱을 포함하여 브라우저의 기능 유형을 검색할 수 있는 방법을 제공하기 위한 것이다. 사용자 에이전트는 반드시 "audio"와 "video"의 종류 값을 지원해야 한다. 시스템에 해당 인수의 값에 해당하는 기능이 없는 경우 getCapability는 null을 리턴한다.

이러한 기능은 일반적으로 장치에 영구적인 cross-origin 정보를 제공하므로 애플리케이션의 핑거프린팅 표면을 증가시킨다. 프라이버시에 민감한 맥락에서 브라우저는 기능의 공통 부분 집합만 보고하는 것과 같은 완화를 고려할 수 있다.

getParameters

getParameters() 메소드는 트랙이 디코딩되는 방법에 대한 RTCRtpReceiver 객체의 현재 파라미터를 반환한다.

getParameters가 호출되면 RTCRtpReceiveParameters 딕셔너리는 다음과 같이 구성된다:

- headerExtensions 시퀀스는 수신자가 현재 수신할 준비가 된 헤더 확장을 기반으로 채워진다.
- codecs는 [[ReceiveCodec]] 내부 슬롯의 값으로 설정된다.

참고

로컬 및 원격 기술 모두 이 코덱 목록에 영향을 미칠 수 있다. 예를 들어, 3개의 코덱이 제공되면, 수신자는 코덱을 각각 수신할 준비가 되어 getParameter로부터 코덱을 모두 반환하게 된다. 그러나 원격 엔드포인트가 2개로만 응답할 경우, 수신기가 더 이상 수신 준비를 할 필요가 없기 때문에 부재중인 코덱은 더 이상 getParameters에 의해 반환되지 않을 것이다.

- 수신자가 현재 축소된 크기의 RTCP 패킷을 수신할 준비가 되어 있는 경우 rtcp.reducedSize는 true로 설정되고, 그렇지 않은 경우 false로 설정된다. rtcp.cname은 제외된다.

getContributingSources

마지막 10초 동안 이 RTCRtpReceiver에 의해 수신된 각 고유 CSRC 식별자에 대한 RTCRtp ContribingSource를 내림차순 타임스탬프 순서로 반환한다.

getSynchronizationSources

마지막 10초 동안 이 RTCRtpReceiver에 의해 수신된 각 고유 SSRC 식별자에 대한 RTCRtp SynchronizationSource를 내림차순 타임스탬프 순서로 반환한다.

getStats

이 수신자에 대한 통계만 수집하고 결과를 비동기식으로 보고한다.
getStats() 메소드가 호출되면 사용자 에이전트는 다음 단계를 실행해야 한다:

1. selector를 메소드가 호출된 RTCRtpReceiver 객체로 둔다.
2. p를 새로운 프로미스로 삼고, 다음 단계를 병렬로 실행한다.
   1) 통계 선택 알고리즘에 따라 selector로 표시된 통계를 수집한다.
   2) 수집된 통계를 포함하는 결과인 RTCStatsReport 객체로 p를 정한다.
3. p를 리턴한다.

RTCRtpContrubritingSource 및 RTCRtpSynchronizationSource 딕셔너리에는 각각 주어진 기여 소스(CSRC) 또는 동기화 소스(SSRC)에 대한 정보가 포함되어 있다. 하나 이상의 RTP 패킷의 오디오 또는 비디오 프레임이 RTCRtpReceiver의 MediaStreamTrack에 전달되면,

사용자 에이전트는 해당 패킷의 내용을 기반으로 RTCRTPContrubritingSource 및 RTCRTP SynchronizationSource 딕셔너리에 대한 관련 정보를 업데이트하는 작업을 대기열에 넣어야 한다.

SSRC 식별자에 해당하는 RTCRtpSynchronizationSource 사전과 관련된 정보는 매번 업데이트되며, RTP 패킷에 CSRC 식별자가 포함된 경우, 해당 CSRC 식별자에 해당하는 RTCRtp ContrublicSource 사전과 관련된 정보도 업데이트된다.

사용자 에이전트는 RTP 타임스탬프의 오름차순으로 RTP 패킷을 처리해야 한다. 사용자 에이전트는 이전 10초 이내에 RTCRtpReceiver의 MediaStreamTrack에 전달된 RTP 패킷의 정보를 반드시(MUST) 보관해야 한다.

참고

MediaStreamTrack이 플레이아웃을 위해 싱크에 연결되어 있지 않더라도 getSynchronization Sources 및 getControvingSources는 트랙이 종료되지 않는 한 최신 정보를 반환하며, 싱크는 RTP 패킷을 디코딩하기 위한 필수 조건이 아니다.

참고

적합성 섹션에 명시한 대로, 최종 결과가 동등한 한 어떠한 방식으로의 알고리즘으로 구현될 수 있다. 따라서 하나의 이벤트 루프 작업 실행 내에서 특정 RTCRTPRTPReceiver를 위한 모든 리턴된 RTCRtpSynchronizationSource 및 RTCRTPContractingSource 딕셔너리가 RTP 스트림의 단일 지점에서 정보를 포함하는 한, 구현은 모든 프레임에 대해 작업을 문자 그대로 대기열에 넣을 필요가 없다.

WebIDL

```
dictionary RTCRtpContributingSource {
 required DOMHighResTimeStamp timestamp;
 required unsigned long source;
 double audioLevel;
 required unsigned long rtpTimestamp;
};
```

RTCRtpContributingSource 딕셔너리 멤버

timestamp : DOMHighResTimeStamp 타입, 요구됨

원본에서 시작된 RTP 패킷의 프레임이 RTCRtpReceiver의 MediaStreamTrack에 전달된 가장 최근의 시간을 나타내는 타임스탬프. 타임스탬프는 당시 Performance.timeOrigin + Performance.now()로 정의된다.

source : unsigned long 타입, 요구됨

기여 또는 동기화 소스의 CSRC 또는 SSRC 식별자

audioLevel : double 타입

오디오 수신자 전용. 이 값은 0..1(선형) 사이의 값이며, 여기서 1.0은 0 dBov를 나타내고, 0은 음소거를 나타내고, 0.5는 0 dBov에서 음압 수준의 약 6 dBSPL 변화를 나타낸다.

CSRC의 경우, RFC 6465 헤더 확장이 있는 경우 [RFC6465]에 정의된 레벨 값에서 변환해야 하며, 그렇지 않은 경우 이 멤버는 반드시(MUST) 존재하지 않아야 한다.

SSRC의 경우, 반드시 [RFC6464]에 정의된 수준 값에서 변환해야 한다. RFC 6464 헤더 확장이 수신된 패킷에 없는 경우(예: 다른 엔드포인트가 사용자 에이전트가 아니거나 레거시 엔드포인트인 경우), 이 값은 없어야 한다.

두 RFC는 시스템이 인코딩할 수 있는 가장 큰 신호에 대해 음의 데시벨로 오디오 레벨을 나타내는 0부터 127까지의 정수 값으로 레벨을 정의한다. 따라서 0은 시스템이 인코딩할 수 있는 가장 큰 신호를 나타내며, 127은 음소거를 나타낸다.

이러한 값을 선형 0..1 범위로 변환하려면 127의 값을 0으로 변환하고, 다른 모든 값은 10^(-rfc_level/20) 방정식을 사용하여 변환한다.

rtpTimestamp : unsigned long 타입, 요구됨

[RFC3550] 섹션 5.1에 정의된 대로 미디어가 재생된 마지막 RTP 타임스탬프.

WebIDL

```
dictionary RTCRtpSynchronizationSource : RTCRtpContributingSource {
 boolean voiceActivityFlag;
};
```

RTCRtpSynchronizationSource 딕셔너리 멤버

voiceActivityFlag : Boolean 타입

오디오 수신자 전용이다. 이 소스에서 전달된 마지막 RTP 패킷이 음성 활동(참) 또는 (거짓)을 포함하는지 여부를 나타낸다. RFC 6464 확장 헤더가 없거나 [RFC6464], 섹션 4, voice ActivityFlag에서 설명한 대로 "vad" 확장 속성을 "off"로 설정하여 피어가 V 비트를 사용하지 않는다는 신호를 보낸 경우이다.

(위험한 기능) 이슈 1

구현자의 명확한 약속이 없기 때문에 voiceActivityFlag는 위험한 기능으로 표시한다.

(4) RTCRtpTransceiver 인터페이스

RTCRtpTransceiver 인터페이스는 공통 미디어 스트림 "identification-tag"를 공유하는 RTCRtpSender와 RTCRtpReceiver의 조합을 나타낸다. [JSEP](섹션 3.4.1)에서 정의한 바와 같이, RTCRtpTransceiver는 "mid" 속성이 null이 아니고 미디어 기술에서 미디어 스트림 "identification-tag"와 일치하는 경우 미디어 기술과 관련이 있다고 하며, 그렇지 않은 경우 해당 미디어 기술과 관련이 없다고 한다.

참고

RTCRtpTransceiver는 JSEP의 새로운 보류 중인 기술과 연결될 수 있으며, 현재 기술과 여전히 관련이 없다. 협상이 필요하면 발생할 수 있는 일이다.

RTCRtpTransceiver의 트랜시버 종류는 연관된 RTCRtpReceiver의 MediaStreamTrack 객체의 종류에 의해 정의된다.

RTCRtpReceiver 객체, receiver, RTCRtpSender 객체, sender 및 RTCRtpTransceiverDirection 값, direction을 사용하여 RTCRtpTransceiverDirection을 생성하려면 다음 단계를 실행한다.

1. transceiver를 새로운 RTCRtpTransceiver 객체로 둔다.
2. transceiver를 sender로 초기화된 [[Sender]] 내부 슬롯을 갖게 한다.
3. transceiver를 receiver로 초기화된 [[Receiver]] 내부 슬롯을 갖게 한다.
4. transceiver를 false로 초기화된 [[Stopping]] 내부 슬롯을 갖게 한다.

5. transceiver를 false로 초기화된 [[Stopped]] 내부 슬롯을 갖게 한다.
6. transceiver를 direction으로 초기화된 [[Direction]] 내부 슬롯을 갖게 한다.
7. transceiver를 false로 초기화된 [[Receptive]] 내부 슬롯을 갖게 한다.
8. transceiver를 null로 초기화된 [[CurrentDirection]] 내부 슬롯을 갖게 한다.
9. transceiver를 null로 초기화된 [[FiredDirection]] 내부 슬롯을 갖게 한다.
10. transceiver를 빈 리스트로 초기화된 [[PreferredCodecs]] 내부 슬롯을 갖게 한다.
11. transceiver를 null로 초기화된 [[JsepMid]] 내부 슬롯을 갖게 한다. 이것은 [JSEP](제5.2.1절 및 제5.3.1절)에 정의된 "RtpTransceiver mid property" 이며, 그곳에서만 수정된다.
12. transceiver를 null로 초기화된 [[Mid]] 내부 슬롯을 갖게 한다.
13. transceiver를 리턴한다.

참고

transceiver를 생성한다고 해서 기본 RTCDtlsTransport 및 RTCCIceTransport 객체가 생성되는 것은 아니다. 이는 RTCSessionDescription을 설정하는 프로세스의 일부로만 발생한다.

WebIDL

```
[Exposed=Window]
interface RTCRtpTransceiver {
 readonly attribute DOMString? mid;
 [SameObject] readonly attribute RTCRtpSender sender;
 [SameObject] readonly attribute RTCRtpReceiver receiver;
 attribute RTCRtpTransceiverDirection direction;
 readonly attribute RTCRtpTransceiverDirection? currentDirection;
 undefined stop();
 undefined setCodecPreferences(sequence<RTCRtpCodecCapability> codecs);
};
```

속성

mid : DOMString 타입, 읽기 전용, null 가능

mid 속성은 로컬 및 원격 기술에 협상되고 표시되는 미디어 스트림 "identification-tag"이다. 가져올 때, 속성은 [[Mid] 슬롯 값을 반환해야 한다.

sender : RTCRtpSender 타입, 읽기 전용

sender 속성은 mid = [[Mid]]로 송신할 수 있는 RTP 미디어에 해당하는 RTCRtpSender를 노출한다. 속성을 가져올 때, 반드시 [[Sender]] 슬롯 값을 반환한다.

receiver : RTCRtpReceiver 타입, 읽기 전용

receiver 속성은 mid = [Mid]로 수신할 수 있는 RTP 미디어에 해당하는 RTCRtp Receiver이다. 속성을 가져올 때, 반드시 [[Receiver]] 슬롯 값을 반환한다.

direction : RTCRtpTransceiverDirection 타입

[JSEP](섹션 4.2.4.)에서 정의한 바와 같이 direction 속성은 이 transceiver의 선호 방향을 나타내며, 이는 offer를 생성하고 응답하기 위한 호출에 사용될 것이다. 방향성 업데이트는 즉시 적용되지 않는다. 대신, 향후 createOffer 및 createAnswer에 대한 호출은 해당 미디어 기술을 [JSEP]에 정의된 대로 sendrecv, sendonly, recvonly 또는 inactive로 표시한다(섹션 5.2.2 및 섹션 5.3.2).

가져올 때, 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야 한다:

1. transceiver를 게터(getter)가 호출되는 RTCRtpTransceiver 객체가 되도록 한다.
2. 만약 transceiver.[[Stopping]] 이 true라면, “stopped”를 리턴한다.
3. 그렇지 않으면, [[Direction]] 슬롯의 값을 리턴한다.

설정할 때, 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야 한다:

1. transceiver를 세터(setter)가 호출되는 RTCRtpTransceiver 객체가 되도록 한다.)
2. 연결을 transceiver와 연관된 RTCPeerConnection 객체로 둔다.
3. transceiver.[[Stopping]]이 true라면, InvalidStateError를 발생시킨다.
4. newDirection을 setter의 인자로 둔다.
5. 만약 newDirection이 transceiver.[[Direction]]과 같다면, 단계를 중단한다.
6. 만약 newDirection이 “stopped”와 같다면, TypeError를 발생시킨다.

7. transceiver.[[Direction]]을 newDirection으로 설정한다.
8. 연결을 위한 negotiation-needed 플래그를 업데이트한다.

currentDirection : RTCRtpTransceiverDirection 타입, 읽기 전용, null 가능

[JSEP](섹션 4.2.5)에서 정의한 바와 같이, currentDirection 속성은 이 트랜시버에 대해 협상된 현재 방향을 나타낸다. currentDirection 값은 RTCRtpEncodingParameters.active의 값과 독립적이다. 이 트랜시버가 offer/answer 교환에 표현된 적이 없는 경우, 그 값은 null이다. 트랜시버가 정지되면 값은 "stopped"가 된다.

가져올 때, 유저 에이전트는 반드시(MUST) 다음 단계를 실행해야 한다:

1. 트랜시버를 게터(getter)가 호출되는 RTCRtpTransceiver 객체가 되도록 한다.
2. 만약 transceiver.[[Stopped]]가 true라면, “stopped”를 리턴한다.
3. 그렇지 않으면, [[CurrentDirecion]] 슬롯의 값을 리턴한다.

메소드

stop

불가역적으로 트랜시버는 이미 정지하지 않은 한 정지된 것으로 표시한다. 이는 즉시 트랜시버의 송신자가 더 이상 송신하지 않고, 트랜시버가 더 이상 수신하지 못하게 할 것이다. Stop()을 호출하는 것 또한 RTCRtpTransceiver의 관련 RTCPeerConnection에 필요한 협상 플래그를 업데이트한다.

트랜시버를 정지시키는 것은 [JSEP](제4.2.1절)에 정의된 바와 같이 해당 트랜시버에 대한 미디어 설명에 포트 0번를 생성하도록 미래의 호출을 야기할 것이다(사용자 에이전트는 반드시 이 경우에만 stopping인 트랜시버를 JSEP의 목적으로 stopped으로 취급해야 한다). 그러나 [BUNDLE]과의 문제를 피하기 위해, stopping이지만 stopped가 아닌 트랜시버는 createAnswer에 영향을 미치지 않는다.

정지된 트랜시버는 [JSEP](제4.2.1절)에 정의된 대로 해당 트랜시버에 대한 미디어 설명에 포트 0번을 생성하기 위해 향후 createOffer 또는 createAnswer를 호출한다.

트랜시버는 원격 offer나 answer에서 거부된 m-line을 setRemoteDescription가 처리하여 stopped 되지 않은 한 stopping 상태를 유지한다.

참고

stopped가 아니라 stopping인 트랜시버는 항상 협상이 필요하다. 실제로 이는 양쪽에서 협상이 완료될 수 있다면 트랜시버의 stop() 메소드가 결국 트랜시버를 정지시키는 원인이 된다는 것을 의미한다.

stop 메소드가 호출되면 사용자 에이전트는 반드시 다음 단계를 실행해야 한다.

1. transceiver를 메소드가 호출되는 RTCRtpTransceiver 객체가 되도록 한다.
2. connection을 transceiver와 관련된 RTCPeerConnection 객체로 둔다.
3. 만약 connection.[[IsClosed]]가 true라면, InvalidStateError를 발생시킨다.
4. 만약 transceiver.[[Stopping]] 이 true라면, 단계를 중단한다.
5. transceiver를 통하여 수신과 송신을 중단한다.
6. connection을 위한 negotiation-needed 플래그를 업데이트 한다.

false를 기본값으로 하고 사라지는 boolean 값을 갖는 transceiver를 송신 및 수신 중지 알고리즘은 다음과 같다:

1. sender를 transceiver.[[Sender]]로 둔다.
2. receiver를 transceiver.[[Receiver]]로 둔다.
3. sender로 미디어를 송신하는 것을 중단한다.
4. [RFC3550]에 명시된 대로 sender가 송신하고 있던 각 RTP 스트림에 대해 RTCP BYE를 전송한다.
5. receiver로 미디어를 수신하는 것을 중단한다.
6. disappear가 거짓인 경우 receiver.[[ReceiverTrack]]을 종료로 두는 단계를 수행한다. 이것은 이벤트를 일으킨다.
7. transceiver.[[Direction]]을 “inactive”로 설정한다.
8. transceiver.[[Stopping]]을 true로 둔다.

false를 기본값으로 하고 사라지는 boolean 값을 갖는 RTCRtpTransceiver를 중지하는 알고리즘은 다음과 같다:

1. 만약 transceiver.[[Stopping]]이 거짓이면, transceiver와 disappear를 통한 송신과 수신을 중지한다.
2. transceiver.[[Stopped]]를 true로 설정한다.
3. transceiver.[[Receptive]]를 false로 설정한다.
4. transceiver.[[CurrentDirection]]을 null로 설정한다.

setCodecPreferences

setCodecPreferences 메소드는 사용자 에이전트가 사용하는 기본 코덱 기본 설정을 재정의한다. createOffer 또는 createAnswer를 사용하여 세션 기술을 생성할 때 사용자 에이전트는 반드시 codecs 인자에 지정된 순서대로 이 RTCRtpTransceiver에 해당하는 미디어 섹션에 대해 표시된 코덱을 반드시(MUST) 사용해야 한다.

이 메소드는 응용 프로그램이 특정 코덱(RTX/RED/FEC 포함)의 협상을 비활성화할 수 있도록 한다. 또한 응용 프로그램이 원격 피어가 전송 목록에 가장 먼저 나타나는 코덱을 선호하도록 할 수 있다.

코덱 환경설정은 이 방법이 다시 호출될 때까지 이 RTCRtpTransceiver를 포함하는 createOffer 및 createAnswer에 대한 모든 호출에 대해 유효하다. 코덱을 빈 시퀀스로 설정하면 코덱 기본 설정이 기본값으로 재설정된다.

참고

코덱에는 SDP의 각 m= 섹션에 나열된 페이로드 유형이 있으며, 페이로드 유형과 코덱 간의 매핑을 정의한다. 이러한 페이로드 유형은 m=video 또는 m=audio 라인에 의해 선호 순서에 따라 참조되며, 협상되지 않은 코덱은 [JSEP]의 섹션 5.2.1에서 정의한 바와 같이 이 목록에서 나타나지 않는다. 이후에 제거된 이전에 협상된 코덱은 m=video 또는 m=audio 라인에서 사라지며, 코덱 페이로드 유형은 향후 제공이나 응답에 재사용되지 않지만, 그 페이로드 유형은 SDP의 페이로드 유형의 매핑에서도 제거될 수 있다.

setCodecPreference로 전달되는 코덱 시퀀스는 RTCRtpSender.getCapability(kind) 또는 RTCRtpReceiver.getCapability(kind)에서 반환되는 코덱만 포함할 수 있으며, 여기서 종류는 메소드가 호출되는 RTCRtpTransceiver의 일종이다. 또한 RTCRtpCodecCapability 딕셔너리 멤버는 수정할 수 없다. 코덱이 이러한 요구 사항을 충족하지 못하면 사용자 에이전트는 반드시(MUST) EvalidModificationError를 발생시켜야 한다.

참고

[SDP]의 권장 사항으로 인해 응답 작성 호출은 코덱 기본 설정의 공통 부분 집합과 오퍼에 나타나는 코덱만 사용해야 한다. 예를 들어 코덱 선호도가 "C, B, A"이지만 코덱 "A, B"만 제공되었다면 answer은 코덱 "B, A"만 포함해야 한다. 그러나 [JSEP](섹션 5.3.1)에서는 제공에 없던 코덱을 추가할 수 있으므로 구현이 다르게 동작할 수 있다.

setCodecPreferences() 메소드가 호출되면 사용자 에이전트는 반드시 다음 단계를 실행해야 한다:

1. transceiver를 이 메소드가 호출된 RTCRtpTransceiver 객체로 설정한다.
2. codecs를 첫 번째 인자로 둔다.
3. 만약 codecs가 빈 리스트라면, transceiver.[[PreferredCodecs]]를 codecs로 두고 단계를 중단한다.
4. codecs에서 중복 값을 제거한다. 코덱의 우선순위가 유지되도록 목록 뒤쪽에서 시작한다. 목록 내에서 코덱이 처음 발생하는 인덱스는 이 단계 전후와 동일하다.
5. kind를 transceiver의 transceiver kind로 둔다.
6. 만약 코덱과 RTCRtpSender.getCapabilitys(kind).codecs 또는 codecs과 RTCRtpReceiver.getCapabilitys(kind).codecs 사이의 교차점이 RTX, RED 또는 FEC 코덱만 포함하거나 빈 집합인 경우 InvalidModificationError를 발생시킨다. 이것은 transceiver.direction에 상관없이 우리가 항상 어떤 것이라도 offer 할 수 있는 것을 보장한다.
7. codecCapabilities를 RTCRtpSender.getCapabilities(kind).codecs과 RTCRtpReceiver.getCapabilities(kind).codecs의 합집합으로 둔다.
8. codecs의 각 codecs에 대하여

1) 만약 ecs가 codecCapabilities에 포함하지 않는다면, InvalidModificationError를 발생시킨다.

9. transceiver.[[PreferredCodecs]]를 codecs로 설정한다.

참고

만약 설정된다면, offerer의 코덱 선호도는 offer에서 코덱의 순서를 결정할 것이다. 만약 응답자가 코덱 선호도를 가지고 있지 않다면, 같은 순서가 답안에 사용될 것이다. 그러나, 응답자가 코덱 기본 설정도 가지고 있다면, 이러한 기본 설정은 answer의 순서보다 우선한다. 이 경우, offerer의 선호도는 어떤 코덱이 제공되는지 최종 순서가 아닌지에 영향을 미칠 것이다.

① 시뮬캐스트 기능

시뮬캐스트 기능은 RTCPeerConnection 객체의 addTransceiver 방법과 RTCRtpSender 객체의 setParameters메소드를 통해 제공된다.

addTransceiver 메소드는 인코딩 순서와 함께 전송될 수 있는 최대 시뮬캐스트 스트림 수를 포함하는 시뮬캐스트 엔벨롭(봉투)을 설정한다. 개별 시뮬캐스트 스트림의 특성은 setParameters 메소드를 사용하여 수정할 수 있지만, 시뮬캐스트 엔벨롭은 변경할 수 없다. 이 모델의 시사점 중 하나는 addTrack() 메소드가 인자로 sendEncoding을 사용하지 않기 때문에 시뮬캐스트 기능을 제공할 수 없으므로, 시뮬캐스트를 송신하도록 RTCRtpTransceiver를 구성할 수 없다는 것이다.

또 다른 함축적 의미는 answerer가 시뮬캐스트 엔벨롭을 직접 설정할 수 없다는 것이다. RTCPerConnection 객체의 setRemoteDescription 메소드를 호출할 때, 시뮬캐스트 엔벨롭은 RTCRtpTransceiver에 지정된 RTCSessionDescription에 의해 설명되는 레이어를 포함하도록 구성된다. 일단 엔벨롭이 결정되면 레이어를 제거할 수 없다. active 멤버를 false로 설정하여 레이어를 효과적으로 비활성화함으로써 비활성 상태로 표시할 수 있다.

setParameter는 시뮬캐스트 엔벨롭을 수정할 수 없지만, 전송되는 스트림의 수와 스트림의 특성을 제어할 수 있다. setParameter를 사용하면 active 멤버를 false로 설정하여 시뮬캐스트 스트림을 비활성화할 수 있으며, active 멤버를 true로 설정하여 다시 활성화할 수 있다. setParameter를 사용하면 maxBitrate와 같은 속성을 수정하여 스트림 특성을 변경할 수 있다.

참고

Simulcast는 여러 인코딩을 SFU로 전송하는 데 자주 사용되며, 이 인코딩은 최종 사용자에게 Simulcast 스트림 중 하나를 전달한다. 따라서 사용자 에이전트는 모든 시뮬캐스트 스트림이 스스로 사용할 수 있도록 인코딩 간에 대역폭을 할당해야 한다. 예를 들어, 두 시뮬캐스트 스트림이 동일한 maxBitrate를 가질 경우 두 스트림에서 유사한 비트 전송률을 보일 것으로 예상한다. 대역폭이 모든 시뮬캐스트 스트림이 사용 가능한 형태로 전송되는 것을 허용하지 않는 경우, 사용자 에이전트는 일부 시뮬레이션 스트림 전송을 중지할 것으로 예상된다.

[JSEP](섹션 3.7)에서 정의한 바와 같이, 사용자 에이전트의 제안은 a=simulcast 라인에 "send" 설명만 포함하고 "recv" 기술은 포함하지 않는다. 대안 및 제한사항([MMUSIC-SIMULCAST]에서 설명함)은 지원되지 않는다.

이 규격은 createOffer, createAnswer 또는 addTransceiver를 사용하여 여러 RTP 인코딩의 수신을 구성하는 방법을 정의하지 않는다.

참고

RTCRtpReceiver는 선택적 전달 장치(SFU)가 사용자 에이전트에서 수신하는 시뮬캐스트 스트림 간에 전환하는 시나리오에서 여러 개의 RTP 스트림을 수신할 수 있다. SFU가 RTP 헤더를 재작성하지 않는 경우, 변환된 스트림을 전달하기 전에 단일 RTP 스트림으로 배열하기 위하여 RTCRtpReceiver는 각각 고유의 SSRC와 시퀀스 번호 공간이 있는 구별되는 RTP 스트림에서 패킷을 수신한다. SFU는 주어진 시간에 오직 하나의 RTP 스트림을 포워드 할 수 있지만, 복수의 RTP 스트림의 패킷은 재주문 때문에 receiver에서 혼합될 수 있다. 따라서 다중 RTP 스트림을 수신하기 위해 장착된 RTCRtpReceiver는 수신된 패킷을 올바르게 정렬하고, 잠재적 손실 이벤트를 인식하고, 이에 대응할 수 있어야 한다. 이 시나리오에서 올바른 작동은 비독점적이므로 이 규격의 구현을 위한 선택 사항이다.

그러나 setRemoteDescription이 [JSEP]에 정의된 대로 복수의 RTP 인코딩을 전송할 수 있는 해당 원격 기술로 호출되고 브라우저가 다중 RTP 인코딩 수신을 지원하는 경우, RTCRtpReceiver는 transceiver의 receiver.getParameters()이 협상한 인코딩을 반영하는 다중 RTP 인코딩과 파라미터들을 받을 수 있다.

ⓐ **인코딩 파라미터 예제**

이 섹션은 비규범적이다.

인코딩 파라미터를 사용하여 구현된 시뮬캐스트 시나리오의 예:

EXAMPLE 3

```
// 최저 해상도 레이어를 비활성화 한 3-레이어 공간 시뮬캐스트의 예
var encodings = [
 {rid: 'q', active: true, scaleResolutionDownBy: 4.0}
 {rid: 'h', active: false, scaleResolutionDownBy: 2.0},
 {rid: 'f', active: false},
];
```

② "Hold" 기능

이 섹션은 비규범적이다.

direction 속성과 replaceTrack 메소드를 함께 사용하면 개발자가 "hold" 시나리오를 구현할 수 있다.

피어에 음악을 보내고 수신된 오디오 렌더링을 중지하려면(music-on-hold):

EXAMPLE 4

```
async function playMusicOnHold() {
 try {
 // 오디오 transceiver와 musicTrack이라는 음악 트랙이 있다고 가정한다.
 await audio.sender.replaceTrack(musicTrack);
 // 수신한 오디오를 음소거한다.
 audio.receiver.track.enabled = false;
 // direction을 send-only로 설정한다.(재협상 필요)
 audio.direction = 'sendonly';
 } catch (err) {
```

```
 console.error(err);
 }
}
```

원격 피어의 “sendonly” offer에 답변하려면:

EXAMPLE 5

```
async function handleSendonlyOffer() {
 try {
 // 먼저 sendonly offer를 적용하고,
 // receiver가 ICE Candidates에 준비되었는지 확실히 한다.
 await pc.setRemoteDescription(sendonlyOffer);
 // 오디오 송신을 중단
 await audio.sender.replaceTrack(null);
 // 미래의 협상을 피하기 위해 우리의 direction을 정렬한다
 audio.direction = 'recvonly';
 // createAnswer를 호출하고 recvonly answer를 보낸다
 await doAnswer();
 } catch (err) {
 // signalling error를 핸들
 }
}
```

음악 전송을 중지하고 마이크에서 캡처한 오디오를 전송하여 수신된 오디오를 렌더링하려면:

EXAMPLE 6

```
async function stopOnHoldMusic() {
 // Assume we have an audio transceiver and a microphone track named micTrack
(오디오 transceiver와 micTrack이라는 마이크 트랙이 있다고 가정한다.)
```

```
 await audio.sender.replaceTrack(micTrack);
 // Unmute received audio (받은 오디오를 음소거 해제)
 audio.receiver.track.enabled = true;
 // Set the direction to sendrecv (requires negotiation) (direction을 sendrecv로 설정 (재협상 필요)
 audio.direction = 'sendrecv';
}
```

원격 피어에 의해 보류가 해제되는 것에 대응하려면:

EXAMPLE 7

```
async function onOffHold() {
 try {
 // Apply the sendrecv offer first, to ensure receiver is ready for ICE candidates. (receiver가 ICE Candidates에 준비되었는지 확실히 하기 위해, sendrecv offer를 먼저 적용한다)
 await pc.setRemoteDescription(sendrecvOffer);
 // Start sending audio (오디오 송신을 시작)
 await audio.sender.replaceTrack(micTrack);
 // Set the direction sendrecv (just in time for the answer) (direction을 sendrecv로 설정한다 (answer에 때를 맞춰서)
 audio.direction = 'sendrecv';
 // Call createAnswer and send a sendrecv answer (createAnswer를 호출하고 sendrecv answer를 보낸다.)
 await doAnswer();
 } catch (err) {
 // handle signaling error (signalling error를 핸들링)
 }
}
```

(5) RTCDtlsTransport 인터페이스

RTCDtlsTransport 인터페이스는 RTCRtpSender 및 RTCRTPReceiver 객체에서 RTP 및 RTCP 패킷을 송수신하는 데이터그램 전송 계층 보안(DTLS) 전송에 대한 정보와 SCTP 패킷 및 수신 데이터 등의 다른 데이터에 대한 애플리케이션 액세스를 허용한다. 특히 DTLS는 기반 전송에 보안을 추가하며, RTCDtlsTransport 인터페이스는 기반 전송과 추가된 보안에 대한 정보에 대한 액세스를 허용한다. RTCDtlsTransport 객체는 setLocalDescription() 및 setRemoteDescription()을 호출한 결과로 생성된다. 각 RTCDtlsTransport 객체는 특정 RTCRtpTransceiver의 RTP 또는 RTCP 구성요소에 대한 DTLS 전송 계층을 나타내며, 그러한 그룹이 [BUNDlE]을 통해 협상된 경우 RTCRtpTransceiverser 그룹을 나타낸다.

참고

기존 RTCRtpTransceiver에 대한 새로운 DTLS 연결은 기존 RTCDtlsTransport 객체로 표현되며, 새로운 객체로 표현되는 것이 아니라 그에 따라 상태가 업데이트된다.

RTCDtlsTransport에는 "new"로 초기화된 [[DtlsTransportState]] 내부 슬롯과 비어있는 리스트로 초기화된 [[RemoteCertificates]] 슬롯이 있다.

근본적인 DTLS 전송에 인증서 유효성 검사 실패 또는 치명적인 경고([RFC5246] 섹션 7.2 참조)와 같은 오류가 발생할 경우 사용자 에이전트는 반드시(MUST) 다음 단계를 실행하는 작업을 대기열에 넣어야 한다.

1. transport를 RTCDtlsTransport 객체로 두어 상태 업데이트 및 오류 알림을 받도록 한다.
2. 만약 transport의 상태가 이미 “failed”라면, 단계를 중단한다.
3. transport.[[DtlsTransportState]]를 “failed”로 설정한다.
4. RTCErrorEvent 인터페이스를 사용하여 error라는 이름의 이벤트(Fire and event)를 발생시킨다. 이 이벤트의 errorDetail 특성은 "dtls-failure" 또는 "fingerprint-fail"로 설정되며, 다른 필드는 RTCErrorDetailType 열거형 설명에 따라 적절히 설정되어야 한다.
5. transport에서 statechange라는 이름을 가진 이벤트를 발생시킨다.

근본적인 DTLS 전송이 다른 이유로 해당 RTCDtlsTransport 객체의 상태를 업데이트해야 하는 경우 사용자 에이전트는 다음 단계를 실행하는 작업을 대기열에 넣어야 한다.

1. transport를 RTCDtlsTransport 객체로 두어 상태 업데이트 알림을 받도록 한다.
2. newState를 새로운 state로 둔다.
3. transport.[[DtlsTransportState]]를 newState로 설정한다.
4. 만약 newState가 연결된 경우 newRemoteCertificate를 원격에서 사용 중인 인증서 체인으로 하고 각 인증서가 DER(Diginary Unified Encoding Rule) [X690]로 인코딩되도록 하고 transport.[[RemoteCertificates]]를 newRemoteCertificate로 설정한다.
5. transport에서 statechange라는 이름을 가진 이벤트를 발생시킨다.

WebIDL

```
[Exposed=Window]
interface RTCDtlsTransport : EventTarget {
 [SameObject] readonly attribute RTCIceTransport iceTransport;
 readonly attribute RTCDtlsTransportState state;
 sequence<ArrayBuffer> getRemoteCertificates();
 attribute EventHandler onstatechange;
 attribute EventHandler onerror;
};
```

속성

iceTransport : RTCIceTransport 타입, 읽기 전용

iceTransport 속성은 패킷을 보내고 받는 데 사용되는 근본적인 transport이다. 근본적인 transport는 여러 활성 RTCDtlsTransport 객체 간에 공유되지 않을 수 있다.

state : RTCDtlsTransportState 타입, 읽기 전용

state 속성은 반드시(MUST) 가져올 때, [[DtlsTransportState]] 슬롯의 값을 리턴해야 한다.

onstatechange : EventHandler 타입

이 이벤트 핸들러의 이벤트 유형은 statechange이다.

onerror : EventHandler 타입

이 이벤트 핸들러의 이벤트 유형은 error이다.

메소드

getRemoteCertificates

Returns the value of [[RemoteCertificates]].

[[RemoteCertificates]]의 값을 리턴.

RTCDtlsTransportState 열거형

WebIDL

```
enum RTCDtlsTransportState {
  "new",
  "connecting",
  "connected",
  "closed",
  "failed"
};
```

열거형	설명
new	DTLS가 아직 협상 전이다.
connecting	DTLS는 보안 연결 협상과 원격 핑거프린트 검증 작업을 진행 중이다.
connected	DTLS는 보안접속 협상을 완료하고 원격 핑거프린트를 검증했다.
closed	close_notify 경보 또는 close() 호출의 결과로 transport가 의도적으로 닫혔다.
failed	오류의 결과로 transport가 실패함(오류 경보 수신 또는 원격 핑거프린트 검증 실패 등)

① RTCDtlsFingerprint 딕셔너리

RTCDtlsFingerprint 딕셔너리에는 [RFC4572]에 설명된 해시 함수 알고리즘과 인증서 핑거프린트가 포함되어 있다.

WebIDL

```
dictionary RTCDtlsFingerprint {
 DOMString algorithm;
 DOMString value;
};
```

RTCDtlsFingerprint 딕셔너리 멤버

algorithm : DOMString 타입

'Hash function Textual Names' 레지스트리 [IANA-HASH-FIGN]에 정의된 해시함수 알고리즘 중 하나.

value : DOMString 타입

[RFC4572] 섹션 5의 '핑거프린트' 구문을 활용하여 표현된 소문자 16진수 문자열의 인증서 핑거프린트 값.

(6) RTCIceTransport 인터페이스

RTCIceTransport 인터페이스는 패킷을 보내고 받는 ICE 전송에 대한 정보에 대한 응용 프로그램 액세스를 허용한다. 특히, ICE는 응용프로그램이 접근하기를 원하는 상태를 포함하는 피어 투 피어 연결을 관리한다. RTCIceTransport 객체는 setLocalDescription() 및 setRemote Description()을 호출한 결과로 생성된다. 기본 ICE 상태는 ICE 에이전트에 의해 관리된다. 따라서 ICE 에이전트가 아래에 설명된 대로 사용자 에이전트에 표시를 제공할 때 RTCIce Transport의 상태가 변경된다. 각 RTCIceTransport 객체는 특정 RTCRtpTransceiver의 RTP 또는 RTCP 구성요소에 대한 ICE 전송 계층을 나타내며, 또는 RTCRtpTransceiversiver 그룹이 [BUNDLE]을 통해 협상된 경우이다.

참고

기존 RTCRtpTransceiver에 대한 ICE 재시작은 기존 RTCIceTransport 객체로 표현되며, 새로운 객체로 표현되는 것이 아니라 그에 따라 상태가 업데이트된다.

ICE 에이전트가 RTCIceTransport의 후보 세대를 수집하기 시작했다고 표시되면 사용자 에이전트는 다음 단계를 실행하는 작업을 대기열에 반드시(MUST) 넣어야 한다:

1. connection을 ICE Agent와 관련된 RTCPeerConnection 객체로 둔다.
2. 만약 connection.[[IsClosed]] 가 true라면, 단계를 중단한다.
3. transport를 후보 모임이 시작된 RTCIceTransport가 되도록 한다.
4. transport.[[IceGathererState]]를 gathering으로 설정한다.
5. transport에서 gatheringstatechange라는 이름을 가진 이벤트를 발생시킨다.
6. connection의 ICE 수집 상태를 업데이트 한다.

ICE 에이전트가 RTCIceTransport에 대한 후보 세대의 수집을 완료하고 해당 후보자들이 애플리케이션에 표면화되면 사용자 에이전트는 다음 단계를 실행하는 작업을 대기열에 반드시(MUST) 넣어야 한다.

1. connection을 ICE Agent와 관련된 RTCPeerConnection 객체로 둔다.
2. 만약 connection.[[IsClosed]]가 true라면, 단계를 중단한다.
3. 후보 수집이 완료되면 transport를 RTCIceTransport가 되도록 한다.
4. newCandidate를 RTCIceCandidate 생성에서의 결과인 sdpMid 그리고 sdpMLine Index가 RTCIceTransport, usernameFragment와 연결된 값인 수집이 완료된 현재 세대의 후보자의 사용자 이름 조각으로 설정되고 candidate가 빈 스트링으로 설정된 새로운 딕셔너리가 되도록 한다.
5. connection에서 icecandidate 라는 이름을 가진 candidate 속성을 newCandidate로 설정한 RTCPeerConnectionIceEvent 인터페이스를 사용하여 이벤트를 발생시킨다.
6. 다른 세대의 후보자가 수집되고 있다면, 단계를 중단한다.

참고

CE 에이전트가 이전 세대의 후보를 여전히 수집하는 동안 ICE 재시작이 시작되는 경우 이 문제가 발생할 수 있다.

7. transport.[[IceGathererState]]를 complete로 설정한다.
8. transport에서 gatheringstatechange라는 이름을 가진 이벤트를 발생시킨다.
9. connection을 위한 ICE 수집 상태를 업데이트한다.

ICE 에이전트가 새로운 ICE 후보가 RTCIceTransport에 사용 가능하다고 표시할 경우, ICE 후보 풀에서 하나를 가져오거나 처음부터 수집하여 사용자 에이전트는 다음 단계를 실행하는 작업을 대기열에 넣어야 한다:

1. candidate를 가능한 ICE 후보자로 둔다.
2. connection을 ICE Agent와 연관된 RTCPeerCOnnection 객체로 둔다.
3. 만약 connection.[[IsClosed]]가 true라면, 단계를 중단한다.
4. 만약 connection.[[PendingLocalDescription]] 또는 connection.[[CurrentLocal Description]] 어느 쪽도 null이 아니고, 후보가 모인 ICE 세대를 대표하면, candidate 및 connection로 후보를 표면화하고, 단계를 중단한다.
5. 그렇지 않으면, candidate를 connection.[[EarlyCandidates]]에 추가한다.)

ICE 에이전트가 [RFC8445] 섹션 7.3.1.1에 따른 역할 충돌과 함께 ICE 바인딩 요청으로 인해 ICE 역할이 변경되었음을 알리는 신호를 보낼 때 UA는 [[IceRole]] 값을 새 값으로 설정하는 작업을 대기열에 넣는다.

연결의 초기 후보를 해제하려면 다음 단계를 실행한다:

1. 각 후보마다, connection.[[EarlyCandidates]]에 있는 candidate에 대해, candidate 및 connection로 후보를 표면화한다.
2. connection.[[EarlyCandidates]]를 빈 리스트로 설정한다.

candidate와 connection으로 후보를 표면화하기 위해서는, 다음 단계를 실행한다:

1. connection.[[IsClosed]]가 true라면, 단계를 중단한다.
2. transport를 후보가 이용할 수 있도록 하는 RTCIceTransport가 되도록 한다.

3. 만약 connection.[[PendingLocalDescription]]이 null이 아니라면, 그리고 후보가 모집된 ICE 세대를 대표한다면, candidate를 connection.[[PendingLocalDescription]].sdp에 추가한다.
4. 만약 connection.[[CurrentLocalDescription]]이 null이 아니라면, 그리고 후보가 모집된 ICE 세대를 대표한다면, candidate를 connection.[[CurrentLocalDescription]].sdp에 추가한다.
5. newCandidate를 RTCIceCandidate 생성에서의 결과인 sdpMid 그리고 sdpMLine Index가 RTCIceTransport, usernameFragment와 연결된 값인 수집이 완료된 현재 세대의 후보자의 사용자 이름 조각으로 설정되고 candidate가 candidate-attribute 문법으로 인코딩된 문자열로 candidate를 대표하기 위하여 설정한다.
6. transport의 로컬 후보자들의 세트에 newCandidate를 추가한다.
7. connection에서 icecandidate라는 이름을 가진 candidate 속성이 newCandidate로 설정된 RTCPeerConnectionIceEvent 인터페이스를 사용하여 이벤트를 발생시킨다.

사용 가능한 연결을 가진 후보 쌍이 발견되어 선택되었기 때문에 RTCIceTransportState의 RTCIceTransportState가 변경되거나 선택된 후보 쌍을 변경하지 않고 변경될 수 있다. 선택한 쌍과 RTCIceTransportState는 관련이 있으며 동일한 작업에서 처리된다.

ICE 에이전트가 RTCIceTransport가 선택한 후보 쌍, RTCIceTransportState 또는 둘 다 변경되었음을 나타내는 경우, 사용자 에이전트는 다음 단계를 실행하는 태스크를 대기열에 넣어야 한다:

1. connection을 ICE Agent와 관련된 RTCPeerConnection 객체로 둔다.
2. 만약 connection.[[IsClosed]]가 true라면, 단계를 중단한다.
3. 상태가 변경중인 RTCIceTransport를 transport로 둔다.
4. selectedCandidatePairChanged를 false로 둔다.
5. transportIceConnectionStateChanged를 false로 둔다.
6. connectionIceConnectionStateChanged를 false로 둔다.
7. Let connectionStateChanged be false. (connectionStateChanged를 false로 둔다.)
8. 만약 transport의 선택된 후보 쌍이 변경되면, 다음 단계를 실행한다:)
   1) newCandidatePair는 표시된 쌍을 나타내는 RTCIceCandidatePair로, 한 쌍을 선택한 경우에는 null로 새로 생성한다.

2) transport.[[SelectedCandidatePair]]를 newCandidatePair로 설정한다.

3) selectedCandidatePairChanged를 true로 설정한다.

9. 만약 transport의 RTCIceTransportState가 변경되면, 다음 단계를 실행한다:

1) transport.[[IceTransportState]]를 새롭게 가리키는 RTCIceTransportState로 설정한다.

2) transportIceConnectionStateChanged를 true로 설정한다.

3) connection의 ICE 연결 상태를 RTCIceConnectionState 열거형에 의해 설명되는 새로운 상태 값을 도출하는 값으로 설정한다.

4) 만약 ice 연결 상태가 전 단계에서 변경되었다면, connectionIceConnectionStateChanged로 true로 설정한다.

5) connection의 연결 상태를 RTCPeerConnectionState 열거형에 의해 설명되는 새로운 상태 값을 도출하는 값으로 설정한다.

6) 만약 연결 상태가 전 단계에서 변경되었다면, connectionStateChanged를 true로 설정한다.

10. 만약 selectedCandidatePairChanged가 true라면, transport에서 selectedcandidatepairchange라는 이름을 가진 이벤트를 발생시킨다.

11. 만약 transportIceConnectionStateChanged가 true라면, transport에서 statechange라는 이름을 가진 이벤트를 발생시킨다.

12. 만약 connectionIceConnectionStateChanged가 true라면, connection에서 iceconnectionstatechange라는 이름을 가진 이벤트를 발생시킨다.

13. 만약 connectionStateChanged가 true라면, connection에서 connectionstatechange라는 이름을 가진 이벤트를 발생시킨다.

RTCIceTransport 객체에는 다음과 같은 내부 슬롯이 있다:

- "new"로 초기화된 [[IceTransportState]]
- "new"로 초기화된 [[IceGathererState]])
- null로 초기화된 [[SelectedCandidatePair]]
- "unknown"으로 초기화된 [[IceRole]]

WebIDL

```
[Exposed=Window]
interface RTCIceTransport : EventTarget {
 readonly attribute RTCIceRole role;
 readonly attribute RTCIceComponent component;
 readonly attribute RTCIceTransportState state;
 readonly attribute RTCIceGathererState gatheringState;
 sequence<RTCIceCandidate> getLocalCandidates();
 sequence<RTCIceCandidate> getRemoteCandidates();
 RTCIceCandidatePair? getSelectedCandidatePair();
 RTCIceParameters? getLocalParameters();
 RTCIceParameters? getRemoteParameters();
 attribute EventHandler onstatechange;
 attribute EventHandler ongatheringstatechange;
 attribute EventHandler onselectedcandidatepairchange;
};
```


속성

role : RTCIceRole 타입, 읽기전용

role 속성은 가져올 때, 반드시(MUST) [[IceRole]] 내부 슬롯의 값을 리턴해야 한다.

component : RTCIceComponent 타입, 읽기 전용

component 속성은 반드시(MUST) transport의 ICE 구성 요소를 반환해야 한다. RTCP mux를 사용할 때, 단일 RTCIceTransport transport는 RTP와 RTCP를 모두 전송하며 component는 "rtp"로 설정된다.

state : RTCIceTransportState 타입의 state, 읽기 전용

state 속성은 가져올 때, 반드시(MUST) [[IceTransportState]] 슬롯 값을 반환해야 한다.

gatheringState : RTCIceGathererState 타입, 읽기 전용

gatheringState 특성은 가져올 때, 반드시 [[IceGatherState]] 슬롯 값을 반환해야 한다.

onstatechange : EventHandler 타입

이벤트 핸들러 타입의 statechange는, RTCIceTransport 상태가 변경될 때마다 반드시 실행되어야 한다.

ongatheringstatechange : EventHandler 타입

이벤트 핸들러 타입의 ongatheringstatechange는, RTCIceTransport의 ICE 모집 상태가 변경될 때마다 반드시(MUST) 실행되어야 한다.

onselectedcandidatepairchange : EventHandler 타입

이벤트 핸들러 타입의 onselectedcandidatepairchange는, RTCIceTransport의 선택된 후보자 쌍이 변경 될 때마다 반드시 실행되어야 한다.

메소드

getLocalCandidates

RTCIceTransport에 대해 수집되어 onicepandidate로 전송된 로컬 ICE 후보를 설명하는 시퀀스를 반환한다.

getRemoteCandidates

addIceCandidate()를 통해 이 RTCIceTransport에서 수신한 원격 ICE 후보를 설명하는 시퀀스를 반환한다.

참고

getRemoteCandidates는 addIceCandidate()를 통해 수신되지 않기 때문에 피어 반사적 후보를 노출하지 않는다.

getSelectedCandidatePair

패킷이 전송되는 선택한 후보 쌍을 반환한다. 이 메소드는 [[SelectedCandidatePair]] 슬롯 값을 반드시 반환해야 한다. RTCIceTransport.state가 "new" 또는 "close"인 경우 get SelectedCandidatePair는 null을 반환한다.

getLocalParameters

RTCIceTransport에서 setLocalDescription을 통해 수신한 로컬 ICE 매개변수를 반환하거나, 아직 수신되지 않은 경우 null을 반환한다.

getRemoteParameters

RTCIceTransport에서 setRemoteDescription을 통해 수신한 원격 ICE 매개 변수를 반환하거나 매개 변수가 아직 수신되지 않은 경우 null을 반환 한다.

① RTCIceParameters 딕셔너리

WebIDL

```
dictionary RTCIceParameters {
 DOMString usernameFragment;
 DOMString password;
};
```

RTCIceParameters 딕셔너리 멤버

usernameFragment : DOMString 타입

[ICE], 섹션 7.1.2.3에 정의된 ICE 사용자 이름 조각.

password : DOMString타입

[ICE], 섹션 7.1.2.3에 정의된 ICE 비밀번호.

② RTCIceCandidatePair 딕셔너리

WebIDL

```
dictionary RTCIceCandidatePair {
 RTCIceCandidate local;
 RTCIceCandidate remote;
};
```

RTCIceCandidatePair 딕셔너리 멤버

local : RTCIceCandidate 타입

로컬 ICE 후보.

remote : RTCIceCandidate 타입

원격 ICE 후보.

③ RTCIceGathererState 열거형

WebIDL

```
enum RTCIceGathererState {
 "new",
 "gathering",
 "complete"
};
```

RTCIceGathererState 열거형 설명	
new	RTCIceTransport는 막 만들어졌고 아직 후보들을 모으기는 시작되지 않았다.
gathering	RTCIceTransport는 후보들을 모으는 과정에 있다.
complete	RTCIceTransport가 수집을 완료했으며 이 전송에 대한 후보 종료 표시가 전송되었다. ICE 재시동으로 인해 다시 시작되기 전까지는 후보를 다시 모으지 않을 것이다.

④ RTCIceTransportState 열거형

WebIDL

```
enum RTCIceTransportState {
  "new",
  "checking",
  "connected",
  "completed",
  "disconnected",
  "failed",
  "closed"
};
```

RTCIceTransportState 열거형 설명	
new	RTCIceTransport는 후보들을 모으거나 원격 후보자들이 공급되기를 기다리고 있으며, 아직 확인을 시작하지 않았다.
checking	RTCIceTransport는 적어도 하나의 원격 후보를 수신하여 후보 쌍을 확인하고 있으며, 아직 연결을 찾지 못하거나 [RFC7675]가 이전에 성공한 모든 후보 쌍에서 실패하였다. 확인 이외에도 여전히 후보가 모이고 있을지도 모른다.
connected	RTCIceTransport는 사용 가능한 연결을 찾았지만, 여전히 더 나은 연결이 있는지 다른 후보 쌍을 확인하고 있다. 또한 여전히 모이거나 추가적인 원격 후보자들을 기다리고 있을 수 있다. 승낙 확인 [RFC7675]이(가) 사용 중인 연결에서 실패하고 사용 가능한 다른 성공적인 후보 쌍이 없는 경우 상태는 "checking"(확인할 후보 쌍이 남아 있는 경우) 또는 "disconnected"(확인할 후보 쌍이 없지만 피어가 여전히 수집 및/또는 추가적인 원격 후보를 대기 중인 경우)
completed	RTCIceTransport는 수집을 마치고, 더 이상 원격 후보가 없다는 표시를 받고, 모든 후보 쌍의 확인을 마치고 연결을 찾았다. 이후 모든 성공적인 후보 쌍에서 승낙 확인[RFC7675]이 확인되면 상태가 "failed"로 전환된다.
disconnected	ICE 에이전트는 이 RTCIceTransport에 대한 연결이 현재 손실된 것으로 결정했다. 이것은 얇은 네트워크에서 간헐적으로 트리거(그리고 행동 없이 스스로 해결)할 수 있는 과도상태다. 이 상태가 결정되는 방법은 구현에 의존한다. 예를 들면 다음과 같다. 사용 중인 연결에 사용되는 네트워크 인터페이스를 잃었다. STUN 요청 응답을 반복적으로 받는 것을 실패한다. 대안으로, RTCIceTransport는 모든 기존 후보 쌍의 확인을 완료하고 연결을 찾지 못했으나(또는 승낙 확인 [RFC7675]이 한 번 성공적이지만, 현재 실패함) 여전히 수집 및/또는 추가 원격 후보를 기다리고 있다.

RTCIceTransportState 열거형 설명	
failed	RTCIceTransport는 수집을 완료했고, 더 이상 원격 후보가 없다는 표시를 받았으며, 모든 후보 쌍의 확인을 완료했으며, 모든 쌍이 연결 점검에 실패했거나 승낙을 잃었다. 이것은 ICE가 다시 시작될 때까지의 영구 상태이다. ICE 재시작으로 인해 연결이 재개될 수 있으므로, "failed" 상태로 들어가더라도 DTLS 전송, SCTP 연결 또는 이 전송을 실행하는 데이터 채널이 닫히거나 트랙이 음소거 상태가 되지 않는다.
closed	RTCIceTransport가 종료되어 STUN 요청에 더 이상 응답하지 않음.

참고

성공적인 통화를 위한 가장 일반적인 전환은 new → checking → connected → completed 이지만, 구체적인 상황에서(마지막으로 체크한 후보만 성공하고, 모이고, 더 이상 성공하기 전에 후보 없음 표시가 모두 일어나는 경우) 국가는 "checking"에서 "completed"로 직접 전환할 수 있다.

ICE 재시작은 후보 수집과 연결성 점검이 새로 시작되어 "completed" 상태에서 시작되면 "connected" 상태로 전환된다. 일시적 "disconnected" 상태에서 시작되면 "checking"으로 전환하여 이전에 연결이 끊겼다는 사실을 사실상 잊어버리게 된다.

"failed" 상태와 "completed" 상태는 추가적인 원격 후보가 없다는 표시를 요구한다. 이는 후보 속성이 빈 문자열로 설정된 후보 값으로 addIceCandidate를 호출하거나 canTrickleIceCandidates를 false로 설정하면 알 수 있다.

상태 전환(state transitions)의 몇 가지 예는 다음과 같다:

- (setLocalDescription 또는 setRemoteDescription의 결과로 RTCIceTransport가 처음 생성됨) : "new"
- ("new", 원격 피어를 수신함): "checking"
- ("checking", 사용 가능한 연결을 찾음): "connected"
- ("checking", 확인을 실패하였지만 수집은 계속 진행중): "disconnected"
- ("checking", 포기함): "failed"
- ("disconnected", 새로운 로컬 후보): "checking"
- ("connected", 모든 확인을 완료함): "completed"
- ("completed", 연결을 잃음): "disconnected"
- ("disconnected" or "failed", ICE 재시작이 일어남): "checking"

- ("completed", ICE 재시작이 일어남 : "connected"
- RTCPeerConnection.close(): "closed"

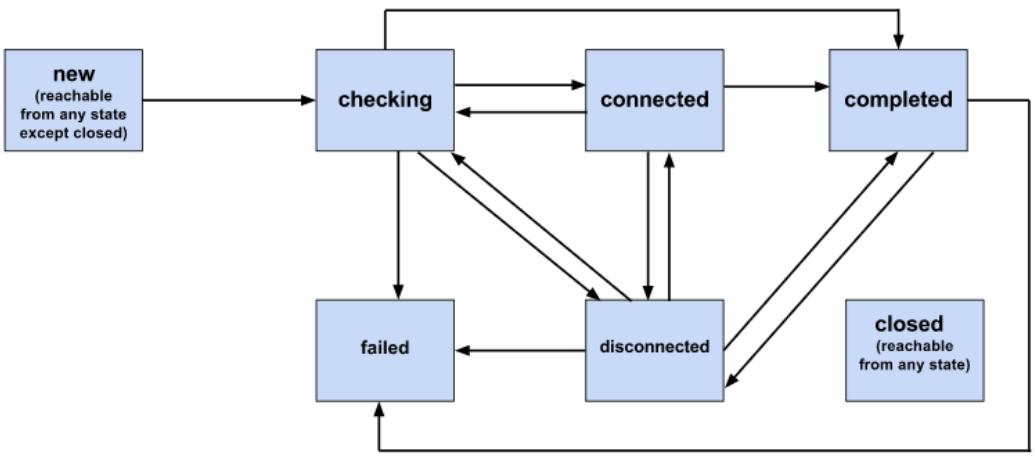


그림 2. 비표준 ICE transport 상태 전환 다이어그램

⑤ RTCIceRole 열거형

WebIDL

```
enum RTCIceRole {
 "unknown",
 "controlling",
 "controlled"
};
```

RTCIceRole 열거형 설명	
unknown	[ICE], 섹션 3에서 정의한 역할이 아직 결정되지 않은 에이전트.
controlling	[ICE], 섹션 3에 정의된 제어 중 에이전트.
controlled	[ICE], 섹션 3에 정의된 제어된 에이전트.

⑥ RTCIceComponent 열거형

WebIDL

```
enum RTCIceComponent {
 "rtp",
 "rtcp"
};
```

RTCIceComponent 열거형 설명	
rtp	ICE transport는 [ICE]에서 정의한 RTP(또는 RTCP 멀티플렉싱)에 사용된다. 섹션 4.1.1.1. RTP(예: 데이터 채널)로 멀티플렉싱된 프로토콜은 구성요소 ID를 공유한다. 이것은 후보 등록으로 인코딩된 경우 component-id 값 1을 나타낸다.
rtcp	ICE 운송은 [ICE], 섹션 4.1.1.1에서 정의한 RTCP에 사용된다. 이것은 후보 등록으로 인코딩되었을 때 component-id 값 2를 나타낸다.

(7) RTCTrackEvent

track 이벤트는 RTCTrackEvent 인터페이스를 사용한다.

WebIDL

```
[Exposed=Window]
interface RTCTrackEvent : Event {
 constructor(DOMString type, RTCTrackEventInit eventInitDict);
 readonly attribute RTCRtpReceiver receiver;
 readonly attribute MediaStreamTrack track;
 [SameObject] readonly attribute FrozenArray<MediaStream> streams;
 readonly attribute RTCRtpTransceiver transceiver;
};
```

생성자

RTCTrackEvent.constructor()

속성

receiver : RTCRtpReceiver 타입, 읽기 전용

receiver 속성은 이벤트와 관련된 RTCRtpReceiver 객체를 나타낸다.

track : MediaStreamTrack 타입, 읽기 전용

track 속성은 receiver로 식별된 RTCRtpReceiver와 연결된 MediaStreamTrack 객체를 나타낸다.

streams : FrozenArray<MediaStream> 타입, 읽기 전용

stream 속성은 이 이벤트의 트랙이 속한 MediaStreams를 나타내는 MediaStream 객체 배열을 반환한다.

transceiver : RTCRtpTransceiver 타입, 읽기 전용

transceiver 속성은 이벤트와 관련된 RTCRtpTransceiver 객체를 나타낸다.

WebIDL

```
dictionary RTCTrackEventInit : EventInit {
 required RTCRtpReceiver receiver;
 required MediaStreamTrack track;
 sequence<MediaStream> streams = [];
 required RTCRtpTransceiver transceiver;
};
```

RTCTrackEventInit 딕셔너리 멤버

receiver : RTCRtpReceiver 타입, 요구됨

receiver 멤버는 이벤트와 관련된 RTCRtpReceiver 객체를 나타낸다.

track : MediaStreamTrack 타입, 요구됨

track 멤버는 receiver로 식별된 RTCRtpReceiver와 연결된 MediaStreamTrack 객체를 나타낸다.

streams : sequence<MediaStream> 타입, 기본값 []

streams 멤버는 이 이벤트의 track이 속한 MediaStreams를 나타내는 MediaStream 객체의 배열이다.

transceiver : RTCRtpTransceiver 타입, 요구됨

transceiver 속성은 이벤트와 관련된 RTCRtpTransceiver 객체를 나타낸다.

6) 피어투피어 데이터 API (Peer-to-Peer Data API)

피어-투-피어 데이터 API를 사용하면 웹 애플리케이션이 일반 애플리케이션 데이터를 피어-투-피어로 보내고 받을 수 있다. 데이터를 보내고 받기 위한 API는 웹 소켓의 동작을 모델링(기본으로) 한다.

(1) RTCPeerConnection 인터페이스 확장(연결 확장)

피어투피어 데이터 API는 아래 설명한 것처럼 RTCPeerConnection 인터페이스를 확장한다.

WebIDL

```
partial interface RTCPeerConnection {
 readonly attribute RTCSctpTransport? sctp;
 RTCDataChannel createDataChannel(USVString label,
              optional RTCDataChannelInit dataChannelDict = {});
 attribute EventHandler ondatachannel;
};
```

속성

sctp : RTCScfpTransport 타입, 읽기전용, null 허용

SCTP는 SCTP 데이터가 송신되고 수신되는 곳에서 운송된다(전송된다). SCTP가 협상되어지지 않는다면 그 값은 널(null)이 된다. 이 속성은 반드시(MUST) 내부 슬롯인 [[SctpTransport]]에 저장된 RTCSctpTransport개체를 반환해야 한다.

ondatachannel은 EventHandler 타입. 이 이벤트 핸들러의 이벤트 유형은 데이터채널임.

메소드

createDataChannel

지정된 레이블을 사용하여 새 RTCDataChannel 개체를 만든다. RTCDataChannelInit 사전은 데이터 안정성과 같은 기본 채널의 속성을 구성하는 데 사용할 수 있다.

createDataChannel 메소드가 호출되면 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야한다.

1. connection이 메소드가 호출되는 RTCPeerConnection 객체가 되도록 한다.
2. 만약 connection. [[IsClosed]]가 true이면 InvalidStateError를 발생시킨다.
3. RTCDataChannel, 채널을 만든다.
4. channel. [[DataChannelLabel]]을 첫 번째 인수의 값으로 초기화한다.
5. 만약 [[DataChannelLabel]]의 UTF-8 표현이 65535바이트보다 길면 TypeError를 발생시킨다.
6. 옵션을 두 번째 인수로 둔다.
7. channel. [[MaxPacketLifeTime]]을 option.maxPacketLifeTime(있는 경우)으로 초기화하고 그렇지 않으면 null로 초기화한다.
8. channel. [[MaxRetransmits]]를 option.maxRetransmits(있는 경우)로 초기화하고 그렇지 않으면 null로 초기화한다.
9. channel. [[Ordered]]를 option.order로 초기화한다.
10. channel. [[DataChannelProtocol]]을 option.protocol로 초기화한다.
11. 만약 [[DataChannelProtocol]]의 UTF-8 표현이 65535바이트보다 길면 TypeError를 발생시킨다.

12. channel. [[Negotiated]]를 option.negotiated로 초기화한다.
13. channel. [[DataChannelId]]가 (option.id값이) 존재하고 [[Negotiated]]가 true이면 option.id의 값으로 초기화하고 그렇지 않으면 null을 초기화한다.

참고

이것이 의미하는 것은 데이터 채널이 대역 내에서 협상되면 id 멤버가 무시된다는 뜻이다; 이것은 의도적인 것이다. 대역 내에서 협상 된 데이터 채널에는 [RTCWEB-DATA-PROTOCOL]에 지정된 대로 DTLS 역할에 따라 선택된 ID가 있어야한다.

14. 만약 [[Negotiated]]가 true이고 [[DataChannelId]]가 null이면 TypeError를 발생시킨다.
15. 만약 [[MaxPacketLifeTime]] 및 [[MaxRetransmits]] 속성이 모두 설정된 경우 (null 아님) TypeError를 발생시킨다.
16. 만약 [[MaxPacketLifeTime]] 또는 [[MaxRetransmits]] 설정이 신뢰할 수 없는 모드를 나타내도록 설정되어 있고 해당 값이 사용자 에이전트가 지원하는 최대 값을 초과하는 경우 값은 사용자 에이전트 최대 값으로 설정되어야한다.
17. 만약 [[DataChannelId]]가 허용되는 최대 ID 인 65534보다 크지 만 여전히 부호 없는 short로 규정되는 65535와 같으면 TypeError를 발생시킨다.
18. 만약 [[DataChannelId]] 슬롯이 null이고 (createDataChannel로 전달되는 ID가 없거나 [[Negotiated]]가 false이기 때문에) SCTP 전송의 DTLS 역할이 이미 협상된 경우 [[DataChannelId]]를 [RTCWEB-DATA-PROTOCOL]에 따라 사용자 에이전트가 생성한 값으로 변경하고 다음 단계로 건너뛴다. 사용 가능한 ID를 생성 할 수 없거나 [[Data ChannelId]] 슬롯의 값이 기존 RTCDataChannel에서 사용 중인 경우 OperationError 예외가 발생한다.

참고

이 단계 후에 [[DataChannelId]] 슬롯이 null이면 RTCSctpTransport 연결 절차 중에 채워진다.

19. 전송을 connection. [[SctpTransport]]로 설정한다.
    만약 [[DataChannelId]] 슬롯이 null이 아닌 경우 전송은 "연결됨" 상태이고 [[Data Channel Id]]가 transport. [[MaxChannels]]보다 크거나 같으면 OperationError를 발생시킨다.
20. 만약 채널이 연결 시 생성된 첫 번째 RTCDataChannel인 경우 연결에 대한 협상 필요 플래그를 업데이트한다.
21. 채널을 반환하고 다음 단계를 병렬로 계속한다.
22. 채널의 관련 기본 데이터 전송을 생성하고 채널의 관련 속성에 따라 구성한다.

① RTCSctpTransport 인터페이스

RTCSctpTransport 인터페이스를 사용하면 애플리케이션이 특정 SCTP 연관에 연결된 SCTP 데이터 채널에 대한 정보에 액세스할 수 있다.

ⓐ 인스턴스 생성

초기 상태 인 initialState로 RTCSctpTransport를 생성하려면 다음 단계를 실행한다.

1. transport를 새 RTCSctpTransport 개체로 지정한다.
2. transport를 전송에 initialState로 초기화된 [[SctpTransportState]] 내부 슬롯을 가지도록 한다.
3. transport에 [[MaxMessageSize]] 내부 슬롯이 있도록 하고 데이터 최대 메시지 크기 업데이트라고 표시된 단계를 실행하여 초기화한다.
4. transport이 null로 초기화된 [[MaxChannels]] 내부 슬롯을 가지게 한다.
5. transport를 반환한다.

ⓑ 최대 메시지 크기 업데이트

RTCSctpTransport의 데이터 최대 메시지크기를 업데이트하려면 아래 단계를 실행한다:

1. transport를 업데이트할 RTCSctpTransport 개체로 지정한다.
2. [SCTP-SDP] (섹션 6)에 설명 된대로 remoteMaxMessageSize를 원격 설명에서 읽은 max-message-size SDP 속성의 값 또는 속성이 누락 된 경우 65536으로 지정한다.

3. canSendSize를 이 클라이언트가 보낼 수 있는 바이트 수(즉, 로컬 전송 버퍼의 크기) 또는 구현이 모든 크기의 메시지를 처리할 수 있는 경우 0이라고 한다.
4. 만약 remoteMaxMessageSize와 canSendSize가 모두 0이면 [[MaxMessageSize]]를 양의 무한대 값으로 설정한다.
5. 그렇지 않고, remoteMaxMessageSize 또는 canSendSize가 0이면 [[MaxMessage Size]]를 둘 중 큰 값으로 설정한다.
6. 그렇지 않으면 [[MaxMessageSize]]를 remoteMaxMessageSize 또는 canSendSize 중 더 작은 값으로 설정한다.

ⓒ 연결된 절차

SCTP 전송(RTCSctpTransport의 SCTP 연결)이 연결되면 다음 단계를 실행한다.

1. transport를 RTCSctpTransport 개체로 지정한다.
2. connection이 전송과 관련된 RTCPeerConnection 객체가 되도록 한다.
3. [[MaxChannels]]를 협상된 수신 및 발신 SCTP 스트림의 최소량으로 설정한다.
4. 각 연결의 RTCDataChannel에 대해 :
   1) channel을 RTCDataChannel 개체로 설정한다.
   2) 만약 channel. [[DataChannelId]]가 null이면 [RTCWEB-DATA-PROTOCOL]에 따라 [[DataChannelId]]를 기본 sctp 데이터 채널에서 생성된 값으로 초기화한다.
   3) 만약 channel. [[DataChannelId]]가 transport. [[MaxChannels]]보다 크거나 같거나 이전 단계에서 ID 할당에 실패한 경우 실패로 인해 채널을 닫습니다. 그렇지 않으면 채널을 열림 상태로 알립니다.
5. 전송시 statechange라는 이벤트를 발생시킨다.

참고

이 이벤트는 채널을 열린 상태로 알리면 열린 이벤트가 시작되기 전에 시작된다. 열기 이벤트는 대기 중인 작업에서 시작된다.

WebIDL

```
[Exposed=Window]interface RTCSctpTransport : EventTarget {
 readonly attribute RTCDtlsTransport transport;
 readonly attribute RTCSctpTransportState state;
 readonly attribute unrestricted double maxMessageSize;
 readonly attribute unsigned short? maxChannels;
 attribute EventHandler onstatechange;
};
```

속성

transport : RTCDtlsTransport 유형의 전송, 읽기 전용

송수신될 데이터 채널에 대한 모든 SCTP 패킷상의 transport

state : RTCSctpTransportState 유형의 상태, 읽기 전용

SCTP 전송의 현재 상태. 얻을 때, 이 속성은 [[SctpTransportState]] 슬롯의 값을 반환해야 한다.

maxMessageSize : 제한되지 않은 double 유형, 읽기 전용

RTCDataChannel의 send () 메소드에 전달할 수 있는 데이터의 최대 크기이다. 속성은 가져올 때 [[MaxMessageSize]] 반드시(MUST) 슬롯의 값을 반환해야한다.

maxChannels : unsigned short 유형, 읽기 전용, null 허용

동시에 사용할 수 있는 RTCDataChannel의 최대 양이다. 속성은 가져올 때 반드시(MUST) [[MaxChannels]] 슬롯의 값을 반환해야한다.

참고

이 속성 값은 SCTP 전송이 "연결됨" 상태가 될 때까지 null이 된다.

onstatechange : EventHandler 유형

이 이벤트 핸들러의 이벤트 유형은 statechange이다.

② RTCSctpTransportState 열거형

RTCSctpTransportState는 SCTP 전송의 상태를 나타낸다.

WebIDL

```
enum RTCSctpTransportState {
  "connecting",
  "connected",
  "closed"
};
```

열거형	설명
connecting 연결중	RTCSctpTransport가 연결을 협상하는 중이다. RTCSctpTransport가 생성될 때 [[SctpTransportState]] 슬롯의 초기 상태이다.
connected 연결됨	연결 협상이 완료되면 [[SctpTransportState]] 슬롯을 "연결됨"으로 업데이트하는 작업이 대기열에 추가된다.
closed (연결이) 닫힘	다음과 같은 경우 [[SctpTransportState]] 슬롯을 "닫힘"으로 업데이트하기 위해 작업이 대기열에 추가된다. • SHUTDOWN 또는 ABORT 청크가 수신된다. • SCTP 연결은 피어 연결을 닫거나 원격 설명을 적용하는 등 의도적으로 닫혔다. • at은 데이터를 거부하거나 SCTP 포트를 변경한다. • 기본 DTLS 연결이 "닫힘"상태로 전환되었다. 마지막 전환은 SCTP 연결에 설정된 DTLS 연결이 필요하다는 사실로 인해 논리적이라는 점에 유의하십시오. [RFC8261] 섹션 6.1은 DTLS를 통한 SCTP가 단일 홈임을 지정하고 있으며 대체 전송으로 전환하는 방법이 이 API에서는 정의되어 있지 않다.

(2) RTC데이터채널 (RTCDataChannel)

RTCDataChannel 인터페이스는 두 피어 간의 양방향 데이터 채널을 나타낸다. RTCDataChannel은 RTCPeerConnection 객체의 팩토리 메소드를 통해 생성된다. 브라우저 간에 전송되는 메시지는 [RTCWEB-DATA] 및 [RTCWEB-DATA-PROTOCOL]에 설명되어 있다.

RTCDataChannel과 연결을 설정하는 방법에는 두 가지가 있다.

첫 번째 방법은 협상 된 RTCDataChannelInit 사전 멤버가 설정되지 않거나 기본값인 false로 설정된 피어 중 하나에서 RTCDataChannel을 간단히 만드는 것이다. 이렇게 하면 대역 내 새 채널을 알리고 다른 피어에서 해당 RTCDataChannel 개체를 사용하여 RTCDataChannelEvent를 트리거한다.

두 번째 방법은 응용 프로그램이 RTCDataChannel을 협상하도록 하는 것이다. 이렇게 하려면, 협상 된 RTCDataChannelInit 사전 멤버가 true이고 같은 id로 설정된 RTCDataChannel 객체를 생성하고, 협상된 RTCDataChannelInit 사전 멤버 세트와 함께 해당 RTCDataChannel을 생성해야(SHOULD) 함을 다른 쪽에 대역 외(out-of-band)(예 : 웹 서버를 통해)를 신호로 전송한다.

이렇게 하면 별도로 생성된 두 개의 RTCDataChannel 개체가 연결된다. 두 번째 방법은 비대칭 속성이 있는 채널을 만들고 일치하는 ID를 지정하여 선언적 방식으로 채널을 만들 수 있도록 한다.

각 RTCDataChannel에는 실제 데이터를 다른 피어로 전송하는 데 사용되는 관련 기본 데이터 전송이 있다. RTCSctpTransport(SCTP 연결 상태를 나타냄)를 사용하는 SCTP 데이터 채널의 경우 기본 데이터 전송은 SCTP 스트림 쌍이다. 주문 전달 설정 및 안정성 모드와 같은 기본 데이터 전송의 전송 속성은 채널이 생성될 때 피어에 의해 구성된다. 채널이 생성된 후에는 채널의 속성을 변경할 수 없다. 피어 간의 실제 유선 프로토콜은 WebRTC 데이터 채널 프로토콜 사양 [RTCWEB-DATA]에 의해 지정된다.

RTCDataChannel은 다양한 안정성 모드에서 작동하도록 구성 할 수 있다. 신뢰할 수 있는 채널은 데이터가 재전송을 통해 다른 피어에서 전달되도록 한다. 신뢰할 수없는 채널은 재전송 횟수를 제한하거나(maxRetransmits) 전송(재전송 포함)이 허용되는 시간(maxPacketLifeTime)을 설정하도록 구성된다. 이러한 속성은 동시에 사용할 수 없으며 그렇게 하면 오류가 발생한다. 이러한 속성을 설정하지 않으면 신뢰할 수 있는 채널이 생성된다.

createDataChannel로 생성되거나 RTCDataChannelEvent를 통해 전달되는 RTCDataChannel은 초기에 반드시(MUST) "연결 중" 상태여야 한다. RTCDataChannel 객체의 기본 데이터 전송이 준비되면 사용자 에이전트는 RTCDataChannel을 개방 상태로 알려야 한다.

① 데이터채널 생성하기

RTCDataChannel을 작성하려면 다음 단계를 실행한다.

1. channel을 새로 만든 RTCDataChannel 개체로 설정한다.
2. 채널에 "연결 중"으로 초기화된 [[ReadyState]] 내부 슬롯이 있도록 한다.
3. 채널이 0으로 초기화된 [[BufferedAmount]] 내부 슬롯을 갖도록 한다.
4. 채널에 [[DataChannelLabel]], [[Ordered]], [[MaxPacketLifeTime]], [[MaxRetransmits]], [[DataChannelProtocol]], [[Negotiated]] 및 [[DataChannelId]]라는 이름의 내부 슬롯이 있도록 한다.
5. 채널을 반환한다.

② 데이터채널을 열린 상태로 알리기(Announcing a data channel as open)

사용자 에이전트가 RTCDataChannel이 열려 있음을 알리는 경우 사용자 에이전트는 반드시 (MUST) 다음 단계를 실행하기 위해 작업을 대기열에 넣어야한다.:

1. 연관된 RTCPeerConnection 객체의 [[IsClosed]] 슬롯이 true이면, 이 단계를 중단한다.
2. channel을 발표할 RTCDataChannel 개체로 설정한다.
3. channel. [[ReadyState]]가 "닫기"또는 "닫힘"이면 이 단계를 중단한다.
4. channel. [[ReadyState]]를 "open"으로 설정한다.
5. open at channel 이벤트를 시작한다.

③ 데이터채널 인스턴스 알리기(Announcing a data channel instance)

기본 데이터 전송이 발표 될 때 (다른 피어가 협상되지 않은 채널을 생성했거나 false로 설정된 경우) 생성 프로세스를 시작하지 않은 피어의 사용자 에이전트는 다음 단계를 실행하기 위해 작업을 대기열에 넣어한다.:

1. connection을 기본 데이터 전송과 관련된 RTCPeerConnection 객체로 지정한다.
2. connection. [[IsClosed]]가 true이면 이 단계를 중단한다.
3. RTCDataChannel, 채널을 만든다.
4. 구성을 WebRTC 데이터 채널 프로토콜 사양 [RTCWEB-DATA-PROTOCOL]에 설명된 기본 데이터 전송을 설정하는 프로세스의 일부로 다른 피어로부터 수신한 정보 번들이라고 가정한다.

5. channel. [[DataChannelLabel]], [[Ordered]], [[MaxPacketLifeTime]], [[Max Retransmits]], [[DataChannelProtocol]] 및 [[DataChannelId]] 내부 슬롯을 구성의 해당 값으로 초기화한다.
6. channel. [[Negotiated]]를 false로 초기화한다.
7. channel. [[ReadyState]]를 "open"으로 설정한다(아직 열기 이벤트를 시작하지 않음).

참고

이를 통해 열기 이벤트가 발생하기 전에 데이터 채널 이벤트 핸들러 내부에서 메시지를 보낼 수 있다.

8. 채널 속성이 연결 시 채널로 설정된 RTCDataChannelEvent 인터페이스를 사용하여 data channel이라는 이벤트를 발생시킨다.
9. 데이터 채널을 공개 상태로 알린다.

④ 종료 절차(Closing procedure)

RTCDataChannel 개체의 기본 데이터 전송은 종료 프로 시저를 실행하여 중단되지 않는 방식으로 종료될 수 있다. 이 경우 사용자 에이전트는 다음 단계를 실행하기 위해 작업을 대기열에 넣어야한다.

1. channel을 기본 데이터 전송이 닫힌 RTCDataChannel 개체로 설정한다.
2. 프로 시저가 channel.close에 의해 시작되지 않은 경우 channel. [[ReadyState]]를 "closing"으로 설정하고 채널에서 closed 이벤트를 발생시킨다.
3. 다음 단계를 병렬로 실행한다.
   1) 현재 보류중인 채널의 모든 메시지 보내기를 완료한다.
   2) 채널의 기본 데이터 전송에 대해 정의 된 종료 절차를 따른다.
      (1) SCTP 기반 전송의 경우 [RTCWEB-DATA], 섹션 6.7을 따른다.
   3) 관련 절차에 따라 채널의 데이터 전송을 종료가 되게 한다.

⑤ 데이터채널이 종료됨을 알림

RTCDataChannel 객체의 기본 데이터 전송이 닫히면 사용자 에이전트는 반드시(MUST) 다음 단계를 실행하기 위해 작업을 대기열에 넣어야한다:

1. channel을 기본 데이터 전송이 닫힌 RTCDataChannel 개체로 설정한다.
2. channel. [[ReadyState]]가 "닫힘"이면 이 단계를 중단한다.
3. channel. [[ReadyState]]를 "closed"로 설정한다.
4. 오류로 인해 전송이 종료된 경우 RTCErrorEvent 인터페이스를 사용하여 errorDetail 속성이 채널에서 "sctp-failure"로 설정된 오류라는 이벤트를 발생시킨다.
5. 채널에서 close라는 이벤트를 시작한다.

⑥ 데이터 채널 생성 시 오류

경우에 따라 사용자 에이전트는 RTCDataChannel의 기본 데이터 전송을 생성하지 못할 수 있다. 예를 들어, 데이터 채널의 ID는 SCTP 핸드 셰이크에서 [RTCWEB-DATA] 구현에 의해 협상된 범위 밖에 있을 수 있다. 사용자 에이전트가 RTCDataChannel의 기본 데이터 전송을 생성할 수 없다고 판단하면 사용자 에이전트는 다음 단계를 실행하기 위해 반드시(MUST) 작업을 대기열에 넣어야 한다.:

1. channel을 사용자 에이전트가 기본 데이터 전송을 만들 수 없는 RTCDataChannel 개체로 설정한다.
2. channel. [[ReadyState]]를 "closed"로 설정한다.
3. 채널에서 errorDetail 속성이 "data-channel-failure"로 설정된 RTCErrorEvent 인터페이스를 사용하여 error라는 이벤트를 발생시킨다.
4. 채널에서 close라는 이벤트를 시작한다.

⑦ 데이터채널에서 메시지수신

RTCDataChannel 메시지가 유형 유형 및 데이터 rawData와 함께 기본 데이터 전송을 통해 수신되면 사용자 에이전트는 다음 단계를 실행하기 위해 반드시(MUST) 작업을 큐에 넣어야한다.:

1. channel을 사용자 에이전트가 메시지를 수신한 RTCDataChannel 개체로 설정한다.
2. connection이 채널과 관련된 RTCPeerConnection 객체가 되도록 한다.
3. channel. [[ReadyState]]가 "open"이 아닌 경우 이러한 단계를 중단하고 rawData를 삭제한다.
4. type 및 channel.binaryType을 켜서 하위 단계를 실행한다.:
   - 만약 type이 rawData가 문자열임을 나타내는 경우 :

- rawData를 UTF-8로 디코딩한 결과를 나타내는 DOMString이 데이터가 되도록 한다.
- 만약 type이 rawData가 바이너리이고 binaryType이 "blob"임을 나타내는 경우 : 데이터를 원시 데이터 소스로 rawData를 포함하는 새 Blob 개체로 설정한다.
- 만약 type이 rawData가 바이너리이고 binaryType이 "arraybuffer"임을 나타내는 경우 :
- 데이터를 원시 데이터 소스로 rawData를 포함하는 새로운 ArrayBuffer 객체로 둔다.

5. 원본 속성이 connection. [[DocumentOrigin]]의 원본 직렬화로 초기화되고 데이터 속성이 채널에서 데이터로 초기화된 MessageEvent 인터페이스를 사용하여 message라는 이벤트를 발생시킨다.

WebIDL

```
[Exposed=Window]
interface RTCDataChannel : EventTarget {
 readonly attribute USVString label;
 readonly attribute boolean ordered;
 readonly attribute unsigned short? maxPacketLifeTime;
 readonly attribute unsigned short? maxRetransmits;
 readonly attribute USVString protocol;
 readonly attribute boolean negotiated;
 readonly attribute unsigned short? id;
 readonly attribute RTCDataChannelState readyState;
 readonly attribute unsigned long bufferedAmount;
 [EnforceRange] attribute unsigned long bufferedAmountLowThreshold;
 attribute EventHandler onopen;
 attribute EventHandler onbufferedamountlow;
 attribute EventHandler onerror;
 attribute EventHandler onclosing;
 attribute EventHandler onclose;
 undefined close();
 attribute EventHandler onmessage;
```

```
  attribute DOMString binaryType;
  undefined send(USVString data);
  undefined send(Blob data);
  undefined send(ArrayBuffer data);
  undefined send(ArrayBufferView data);
};
```

속성

Label : USVString 유형, 읽기 전용

abel 속성은 이 RTCDataChannel 개체를 다른 RTCDataChannel 개체와 구별하는 데 사용할 수 있는 레이블을 나타낸다. 스크립트는 동일한 레이블로 여러 RTCDataChannel 객체를 생성할 수 있다. 가져올 때 속성은 반드시(MUST) [[DataChannelLabel]] 슬롯의 값을 반환해야한다.

ordered : 부울 유형, 읽기 전용

ordered 속성은 RTCDataChannel이 주문된 경우 true를 반환하고 비순차적 배송이 허용되는 경우 false를 반환한다. 가져올 때 속성은 반드시(MUST) [[Ordered]] 슬롯의 값을 반환해야한다.

maxPacketLifeTime : unsigned short 유형, 읽기 전용, null 허용

maxPacketLifeTime 속성은 전송 및 재전송이 신뢰할 수 없는 모드에서 발생할 수 있는 기간(밀리 초)의 길이를 리턴한다. 얻을 때 속성은 반드시(MUST) [[MaxPacketLifeTime]] 슬롯의 값을 반환해야한다.

maxRetransmits : unsigned short 유형, 읽기 전용, null 허용

maxRetransmits 속성은 신뢰할 수 없는 모드에서 시도된 최대 재전송 횟수를 반환한다. 가져올 때 속성은 반드시(MUST) [[MaxRetransmits]] 슬롯의 값을 반환해야 한다.

protocol : USVString 유형, 읽기 전용

프로토콜 속성은 이 RTCDataChannel과 함께 사용되는 하위 프로토콜의 이름을 반환한다. 얻을 때 속성은 반드시(MUST) [[DataChannelProtocol]] 슬롯의 값을 반환해야한다.

negotiated : 부울 유형, 읽기 전용

negotiated 속성은 이 RTCDataChannel이 응용 프로그램에 의해 협상된 경우 true를 반환하고 그렇지 않으면 false를 반환한다. 얻을 때 속성은 반드시(MUST) [[협상]] 슬롯의 값을 반환해야한다.

id : unsigned short 유형, 읽기 전용, null 허용

id 속성은 이 RTCDataChannel의 ID를 리턴한다. 값은 초기에 null이며 채널 생성 시 ID가 제공되지 않았고 SCTP 전송의 DTLS 역할이 아직 협상되지 않은 경우 반환된다. 그렇지 않으면 스크립트에서 선택했거나 [RTCWEB-DATA-PROTOCOL]에 따라 사용자 에이전트에서 생성한 ID를 반환한다. ID가 널이 아닌 값으로 설정되면 변경되지 않는다. 가져올 때 속성은 반드시(MUST) [[DataChannelId]] 슬롯의 값을 반환해야 한다.

readyState : RTCDataChannelState 유형, 읽기 전용

readyState 속성은 RTCDataChannel 개체의 상태를 나타낸다. 가져올 때 속성은 반드시(MUST) [[ReadyState]] 슬롯의 값을 반환해야한다.

bufferedAmount : unsigned long, 읽기 전용

bufferedAmount 속성은 획득 시 반드시(MUST) [[BufferedAmount]] 슬롯의 값을 반환해야 한다. 이 속성은 send ()를 사용하여 큐에 넣은 애플리케이션 데이터(UTF-8 텍스트 및 이진 데이터)의 바이트 수를 표시한다. 데이터 전송이 병렬로 발생할 수 있지만, 현재 작업이 경합 상태를 방지하기 위해 이벤트 루프에 다시 양보되기 전에 반환된 값을 줄여서는 안 된다(MUST NOT). 이 값에는 프로토콜에서 발생하는 프레이밍 오버 헤드나 운영 체제 또는 네트워크 하드웨어에서 수행한 버퍼링이 포함되지 않는다.

[[BufferedAmount]] 슬롯의 값은 [[ReadyState]] 슬롯이 "open"인 한 send () 메소드를 호출할 때마다 증가한다. 그러나 채널이 닫히면 슬롯은 0으로 재설정되지 않는다.

기본 데이터 전송이 큐에서 데이터를 보낼 때 사용자 에이전트는 반드시(MUST) 전송 된 바이트 수로 [[BufferedAmount]]를 줄이는 작업을 큐에 넣어야한다.

bufferedAmountLowThreshold : unsigned long 유형

bufferedAmountLowThreshold 속성은 bufferedAmount가 낮은 것으로 간주되는 임계 값을 설정한다. bufferedAmount가이 임계 값 위에서 같거나 아래로 감소하면 bufferedamountlow 이벤트가 발생한다. bufferedAmountLowThreshold는 처음에는 각각의 새 RTCData Channel에서 0이지만 응용 프로그램은 언제든지 해당 값을 변경할 수 있다.

onopen : EventHandler 유형

이 이벤트 핸들러의 이벤트 유형은 open이다

onbufferedamountlow : EventHandler 유형

이 이벤트 핸들러의 이벤트 유형은 bufferedamountlow이다.

onerror : 이벤트핸들러 타입

이 이벤트 핸들러의 이벤트 유형은 RTCErrorEvent이다. errorDetail에는 "sctp-failure"가 포함되고 sctpCauseCode에는 SCTP 원인 코드 값이 포함되며 메시지에는 추가 텍스트와 함께 SCTP 원인별 정보가 포함된다.

onclosing : EventHandler 타입

이 이벤트 핸들러의 이벤트 유형은 closing이다.

onclose : EventHandler타입

이 이벤트 핸들러의 이벤트유형은 close이다.

onmessage : 이벤트 핸들러 유형

이 이벤트 핸들러의 이벤트유형은 message이다.

binaryType : DOMString 타입

binaryType 속성은 가져올 때 반드시(MUST) 마지막으로 설정된 값을 반환해야 한다. 설정 시 새 값이 문자열 "blob" 또는 문자열 "arraybuffer"인 경우 IDL 속성을 이 새 값으로 설정한다. 그렇지 않으면 SyntaxError가 발생한다. RTCDataChannel 객체가 생성될 때 binaryType 속성은 문자열 "blob"로 초기화되어야한다.

이 속성은 바이너리 데이터가 스크립트에 노출되는 방식을 제어한다. 웹 소켓의 binary Type을 참조.

메소드

close

RTCDataChannel을 닫습니다. RTCDataChannel 객체가 이 피어 또는 원격 피어에 의해 생성되었는지 여부에 관계없이 호출될 수 있다.

close 메소드가 호출되면 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야 한다.:

1. 채널이 닫히려고 하는 RTCDataChannel 객체가 되도록 한다.
2. channel. [[ReadyState]]가 "closing" 또는 "closed"이면 이 단계를 중단한다.
3. channel. [[ReadyState]]를 "closing"으로 설정한다.
4. 폐쇄 절차가 아직 시작되지 않은 경우 시작한다.

참고

```
//{{역자주: 아래 4개의 send()는 파라메터별 method() 4개인 경우임
 undefined send(USVString data);
 undefined send(Blob data);
 undefined send(ArrayBuffer data);
 undefined send(ArrayBufferView data);
//}}
```

send (USVString data)

인수 유형 문자열 객체를 사용하여 send () 알고리즘에 설명된 단계를 실행한다.

send(Blob data)

인수 유형 Blob 객체를 사용하여 send () 알고리즘에 설명된 단계를 실행한다.

send (ArrayBuffer data)

인수 유형 ArrayBuffer 객체를 사용하여 send () 알고리즘에서 설명하는 단계를 실행한다.

send(ArrayBufferView data)

인수 유형 ArrayBufferView 객체를 사용하여 send () 알고리즘에 설명된 단계를 실행한다.

send () 메소드는 다른 데이터 인수 유형을 처리하기 위해 오버로드 된다. 메소드의 버전이 호출되면 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야한다.

1. channel을 데이터가 전송될 RTCDataChannel 개체로 설정한다.
2. 만약 channel. [[ReadyState]]가 "open"이 아닌 경우 InvalidStateError를 발생시킨다.
3. 메소드 인수의 유형에 해당하는 하위 단계를 실행한다.

1) 문자열 객체 : 메소드의 인수를 UTF-8로 인코딩 한 결과를 나타내는 바이트 버퍼가 데이터가 되도록 한다.

2) Blob 객체 : 데이터를 Blob 개체가 나타내는 원시 데이터로 둔다.

참고

Blob 개체에서 실제 데이터 검색이 비동기식으로 발생할 수 있지만 사용자 에이전트는 send 메소드가 호출된 순서와 동일한 순서로 채널의 기본 데이터 전송에 데이터를 대기열에 넣는다. 데이터의 바이트 크기는 동기적으로 알려야한다.

3) ArrayBuffer 객체 : data를 ArrayBuffer 객체가 설명하는 버퍼에 저장된 데이터라고 한다.

4) ArrayBufferView 객체 : data는 ArrayBufferView 객체가 참조하는 ArrayBuffer 객체가 설명하는 버퍼의 섹션에 저장된 데이터가 된다.

참고

이 메소드가 오버로드되지 않은 모든 데이터 인수 유형은 TypeError를 발생시킨다. 여기에는 null 및 undefined가 포함된다.

4. 만약 데이터의 바이트 크기가 채널의 연결된 RTCSctpTransport에서 maxMessageSize 값을 초과하면 TypeError를 발생시킨다.
5. 채널의 기본 데이터 전송에서 전송을 위해 데이터를 큐에 넣는다. 사용 가능한 버퍼 공간이 부족하여 데이터를 대기열에 넣을 수 없는 경우 OperationError를 발생시킨다.

참고

실제 데이터 전송은 병렬로 발생합니다. 데이터 전송으로 인해 SCTP 수준 오류가 발생하면 응용 프로그램에 onerror를 통해 비동기 적으로 알림이 전송된다.

6. [[BufferedAmount]] 슬롯의 값을 데이터의 바이트 크기만큼 늘린다.

WebIDL

```
dictionary RTCDataChannelInit {
 boolean ordered = true;
 [EnforceRange] unsigned short maxPacketLifeTime;
 [EnforceRange] unsigned short maxRetransmits;
 USVString protocol = "";
 boolean negotiated = false;
 [EnforceRange] unsigned short id;
};
```

사전 RTCDataChannelInit 멤버

ordered : boolean 유형, 기본값 true

만약 false로 설정하면 데이터가 순서 없이 전달될 수 있다. 기본값인 true는 데이터가 순서대로 전달되도록 한다.

maxPacketLifeTime : unsigned short 유형

승인되지 않은 경우 채널이 데이터를 전송하거나 재전송하는 시간(밀리 초)을 제한한다. 이 값은 사용자 에이전트에서 지원하는 최대 값을 초과하는 경우 고정될 수 있다.

maxRetransmits : unsigned short 유형

성공적으로 전달되지 않은 경우 채널이 데이터를 재전송하는 횟수를 제한한다. 이 값은 사용자 에이전트에서 지원하는 최대 값을 초과하는 경우 고정될 수 있다.

protocol : USVString 유형, 기본값 ""

이 채널에 사용되는 서브 프로토콜 이름이다.

negotiated : 부울 유형, 기본값 false

기본값인 false는 사용자 에이전트에게 대역 내 채널을 알리고 다른 피어가 해당 RTCData Channel 개체를 디스패치하도록 지시한다. true로 설정하면 채널을 협상하고 다른 피어에서 동일한 ID를 가진 RTCDataChannel 개체를 만드는 것은 응용 프로그램에 달려 있다.

참고

true로 설정된 경우 애플리케이션은 다른 피어가 메시지를 수신할 데이터 채널을 생성할 때까지 메시지를 보내지 않도록 주의해야 한다. 연관된 데이터 채널이 없는 SCTP 스트림에서 메시지를 수신하는 것은 정의되지 않은 동작이며 자동으로 삭제될 수 있다. 첫 번째 오퍼 / 답변 교환이 완료되기 전에 두 엔드 포인트가 데이터 채널을 생성하는 한 이것은 불가능하다.

id : unsigned short 유형

협상이 true이면 채널 ID를 설정한다. 협상이 false이면 무시된다.

WebIDL

```
enum RTCDataChannelState {
  "connecting",
  "open",
  "closing",
  "closed"
};
```

RTCDataChannelState 열거 형 설명	
connecting	사용자 에이전트가 기본 데이터 전송을 설정하려고 한다. 이것은 createDataChannel로 생성되었거나 RTCDataChannelEvent의 일부로 전달되었는지 여부에 관계없이 RTCDataChannel 객체의 초기 상태이다.
open	데이터 전송이 설정되고 통신이 가능하다.
closing	데이터 전송을 종료하는 절차가 시작되었다.
closed	데이터 전송이 닫혔거나 설정할 수 없다.

(3) RTC데이터채널이벤트(RTCDataChannelEvent)

datachannel 이벤트는 RTCDataChannelEvent 인터페이스를 사용한다.

WebIDL

```
[Exposed=Window]
interface RTCDataChannelEvent : Event {
 constructor(DOMString type, RTCDataChannelEventInit eventInitDict);
 readonly attribute RTCDataChannel channel;
};
```

생성자

RTCDataChannelEvent.constructor()

속성

channel : RTCDataChannel 유형의 채널, 읽기 전용

채널 속성은 이벤트와 연관된 RTCDataChannel 객체를 나타낸다.

WebIDL

```
dictionary RTCDataChannelEventInit : EventInit {
 required RTCDataChannel channel;
};
```

RTCDataChannelEventInit 사전 멤버

channel : RTCDataChannel 유형, 필수

이벤트를 통해 선언되는 RTCDataChannel 개체이다.

(4) 가비지 컬렉션

RTCDataChannel 객체는 다음과 같은 경우 가비지가 수집되지 않아야합니다.(MUST NOT)

- [[ReadyState]] 슬롯이 "연결 중"이고 열기 이벤트, 메시지 이벤트, 오류 이벤트, 닫기 이벤트 또는 닫기 이벤트에 대해 하나 이상의 이벤트 리스너가 등록되어 있다.
- [[ReadyState]] 슬롯이 "열림"이고 메시지 이벤트, 오류 이벤트, 닫기 이벤트 또는 닫기 이벤트에 대해 하나 이상의 이벤트 리스너가 등록되어 있다.
- [[ReadyState]] 슬롯이 "닫는 중"이고 오류 이벤트 또는 닫기 이벤트에 대해 하나 이상의 이벤트 리스너가 등록되어 있다.
- 기본 데이터 전송이 설정되고 데이터가 전송될 대기열에 추가된다.

7) 피어 투 피어 DTMF

이 섹션에서는 RTCRtpSender .NET을 통해 DTMF(전화 키패드) 값을 보내기 위한 인터페이스에 대해 설명한다 RTCPeerConnection. DTMF가 상대방에게 어떻게 전송되는지에 대한 자세한 내용은 [RTCWEB-AUDIO]에 설명되어 있다.

(1) RTCRtpSender Interface Extensions

피어투피어 DTMF API는 아래와 같이 RTCRtpSender 인터페이스를 확장한다.

WebIDL

```
partial interface RTCRtpSender {
 readonly attribute RTCDTMFSender? dtmf;
};
```

속성

dtmf : RTCDTMFSender 유형, 읽기전용, null 허용

dtmf 어트리뷰트는 DTMF나 지정이 안 됐다면 널인 값을 대표하는(represent) RTCD TMFSender [[Dtmf]] 내부 슬롯의 값을 반환한다. [[Dtmf]] 내부 슬롯은 RTCRtpSender의 [[SenderTrack]]의 종류가 “audio”일 때 설정(set)된다.

(2) RTCDTMFSender

RTCDTMFSender를 생성하려면 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야 한다.

1. dtmf를 새로 생성된 RTCDTMFSender 객체로 설정한다.
2. dtmf가 [[Duration]] 내부 슬롯을 가지게 한다.
3. dtmf가 [[InterToneGap]] 내부 슬롯을 가지게 한다.
4. dtmf가 [[ToneBuffer]] 내부 슬롯을 가지게 한다.

WebIDL

```
[Exposed=Window]
interface RTCDTMFSender : EventTarget {
 undefined insertDTMF(DOMString tones, optional unsigned long duration = 100,
optional unsigned long interToneGap = 70);
 attribute EventHandler ontonechange;
 readonly attribute boolean canInsertDTMF;
 readonly attribute DOMString toneBuffer;
};
```

속성

ontonechange : EventHandler 유형

이 이벤트 핸들러의 이벤트 유형은 tonechange이다.

canInsertDTMF : Boolean 유형, 읽기 전용

RTCDTMFSender dtmfSender 가 DTMF를 보낼 수 있는지 여부. 가져올 때, 사용자 에이전트는 반드시(MUST) dtmfSender에 대해 DTMF를 보낼 수 있는지 결정하는 실행 결과를 반환해야 한다.

toneBuffer : DOMString 유형, 읽기전용

toneBuffer 속성은 반드시(MUST) 재생할 남은 tone의 목록을 반환해야 한다. 이 목록의 구문, 내용 및 해석은 insertDTMF를 보라.

메소드

insertDTMF

RTCDTMFSender 개체의 insertDTMF 메소드는 DTMF 톤을 전송하는 데 사용된다.

톤 매개 변수는 일련의 문자로 취급된다. 0 ~ 9, A ~ D, # 및 * 문자는 관련 DTMF 톤을 생성한다. a~d 문자는 반드시(MUST) 대문자로 정규화 되어야 한다. [RTCWEB-AUDIO] 섹션 3에서 참고로 얘기한 바와 같이, 0~9, A~D, #, "*" 문자에 대한 지원은 필수이다. 문자 "," 는 반드시(MUST) 지원되어야 하며, 톤 매개변수에서 다음 문자를 처리하기 전에 2초의 지연을 표시한다. 다른 모든 문자는 반드시(MUST) 인식되지 않는 것으로 고려해야 한다.

duration 매개 변수는 톤 매개 변수에 전달된 각 문자에 사용할 기간(ms)을 나타낸다. duration은 6000ms를 초과하거나 40ms 미만일 수 없다. 기본 지속 시간은 각 톤에 대해 100ms이다. interToneGap의 파라미터는 MS의 톤 사이의 간격을 나타낸다. 사용자 에이전트는 이를 최소 30ms에서 최대 6000ms로 제한한다. 기본값은 70ms이다.

브라우저는 DTMF가 RTP 패킷의 경계와 일치하도록 시작과 중지를 할 수 있도록 duration과 interToneGap 시간을 증가시킬 수 있다(MAY). 그러나 단일 RTP 오디오 패킷의 duration 이상으로 둘중 하나를 증가해서는 안된다(MUST NOT).

insertDTMF() 메소드가 호출되면, 사용자 에이전트는 반드시(MUST) 다음 과정을 실행해야 한다.

1. sender를 DTMF를 전송하는 데 사용되는 RTCRtpSender로 설정한다.
2. transceiver를 sender와 관련 있는 RTCRtpTransceiver 객체로 설정한다.
3. dtmf를 sender와 연관 있는 RTCDTMFSender로 둔다.
4. 만약 dtmf를 위해 DTMF를 보낼 수 있는지 여부가 false를 반환하면, InvalidStateError를 발생한다.
5. tone s를 메소드의 첫 번째 인자로 한다.
6. duration을 메소드의 두 번째 인자로 한다.
7. interToneGap을 메소드의 세 번째 인자로 한다.
8. 만약 tone에 인식할 수 없는 문자가 포함된 경우 InvalidCharacterError를 발생시킨다.
9. 개체의 [[ToneBuffer]] 슬롯을 tones로 설정한다.
10. dtmf.[[Duration]]을 duration 값으로 설정한다.
11. dtmf. [[InterToneGap]]을 interToneGap 값으로 설정한다.

12. 만약 duration 값이 40ms미만이면 dtmf.[[Duration]]을 40ms로 설정한다.
13. 만약 duration 매개 변수의 값이 6000ms보다 크면 dtmf.[[Duration]]을 6000ms로 설정한다.
14. 만약 interToneGap 값이 30ms미만이면 dtmf.[[InterToneGap]]을 30ms로 설정한다.
15. 만약 interToneGap 값이 6000ms보다 크면 dtmf.[[InterToneGap]]을 6000ms로 설정한다.
16. 만약 [[ToneBuffer은]] 슬롯이 빈 문자열이면, 이 과정을 중지한다.
17. 만약 재생 작업이 실행되도록 예약된 경우, 다음 단계를 중단한다. 그렇지 않으면 다음 단계를 실행하는 작업을 대기열에 넣는다. (Playout task):
    1) 만약 transceiver.[[CurrentDirection]]가 " sendrecv"나 " sendonly" 가 아니라면, 이 단계를 중단한다.
    2) 만약 [[ToneBuffer]] 슬롯이 비어있는 문자열을 포함한다면, RTCDTMFSender 객체에 tone 속성을 빈 문자열로 설정하고 RTCDTMFToneChangeEvent 인터페이스를 사용하는 tonechange 이벤트를 발생시키고, 다음 단계를 중지한다.
    3) [[ToneBuffer]] 슬롯 에서 첫 번째 문자를 제거하고 해당 문자가 tone이 되도록 한다.
    4) 만약 tone이 “,”인 경우, 연결된 RTP 미디어 스트림에서 2000ms 동안 톤 전송을 지연하고, 지금부터 2000ms 후에 실행되는 playout task 라벨된 단계를 대기열에 넣는다.
    5) 만약 tone이 ","이 아니라면, (적절한 코덱을 사용하여) 연관된 RTP 미디어 스트림에 대해 [[Duration]]ms 동안 재생을 시작하고, (Playout tast 라벨링된 단계를 실행하는) [[Duration]]+[[InterToneGap]]ms 작업 대기열을 실행한다.
    6) RTCDTMFSender 객체에 설정된 tone 속성의 RTCDTMFToneChangeEvent 인터페이스를 사용하는 tonechange라 명명된 이벤트를 실행한다.

insertDTMF톤 버퍼를 대체하기 때문에 재생중인 DTMF 톤을 추가 insertDTMF하려면 나머지 톤 ([[ToneBuffer]] 슬롯에 저장 됨)과 함께 추가된 새 톤 을 모두 포함하는 문자열로 호출해야 한다. insertDTMF빈 톤 매개 변수를 사용하여 호출하면 현재 재생 중인 톤 이후 재생 대기 중인 모든 톤을 취소 할 수 있다.

(3) canInsertDTMF 알고리즘

DTMF가 RTCDTMFSender 인스턴스인 dtmfSender를 위해 송신될 수 있는지 결정하기 위해, 사용자 agent는 반드시(MUST) 다음과 같은 단계를 수행하는 작업을 대기시켜야 한다.

1. sender를 dtmfSender와 관련된 RTCRtpSender로 설정한다.
2. transceiver를 sender와 연관된 RTCRtpTransceiver로 설정한다.
3. connection을 transceiver와 연관된 RTCPeerConnection으로 설정한다.
4. 만약 connction의 RTCPeerConnectionState가 송신자로 “connected”가 아니라면 false를 리턴한다.
5. 만약 sender.[[SenderTrack]]이 null인 경우 false를 리턴한다.
6. 만약 transceiver.[[CurrentDirection]]이 “sendrecv” 도 아니고 “sendonly”도 아닌 경우 false를 리턴한다.
7. 만약 sender.[[SendEncodings]][0].active가 false이면 false를 리턴한다.
8. 만약 이 sender로 전송할 “audio/telephone-event” mimetype의 codec이 협상(negotiated) 되지 않았다면, false를 반환한다.
9. 그렇지 않으면 true를 반환한다.

(4) RTCDTMFToneChangeEvent

tonechange 이벤트는 RTCDTMFToneChangeEvent 인터페이스를 사용한다.

WebIDL

```
[Exposed=Window]
interface RTCDTMFToneChangeEvent : Event {
 constructor(DOMString type, optional RTCDTMFToneChangeEventInit eventInitDict
= {});
 readonly attribute DOMString tone;
};
```

생성자

RTCDTMFToneChangeEvent.constructor()

속성

tone : DOMString 유형, 읽기 전용

tone속성은 재생을 시작한 톤(“,”를 포함)의 특성을 가지고 있다(insertDTMF참고). 값이 빈 문자열이면 [[ToneBuffer]] 슬롯이 빈 문자열이고 이전 톤이 재생을 완료했음을 나타낸다.

WebIDL

```
dictionary RTCDTMFToneChangeEventInit : EventInit {
 DOMString tone = "";
};
```

RTCDTMFToneChangeEventInit 사전 멤버

tone : DOMString 유형, 기본값 ""

tone속성은 재생을 시작한 톤(“,”를 포함)의 특성을 가지고 있다 (insertDTMF참고). 값이 빈 문자열이면 [[ToneBuffer]] 슬롯이 빈 문자열이고 이전 톤이 재생을 완료했음을 나타낸다.

8) 통계 모델

(1) 소개

기본적인 통계 모델은 브라우저가 통계 객체의 형태로 모니터링 객체에 대한 통계 집합을 유지하는 것이다.

관련 객체 그룹은 선택자(selector)에 의해 참조될 수 있다. 예를 들어, 선택자(selector)는 MediaStreamTrack일 수 있다. 트랙이 유효한 선택자(selector)가 되려면, 반드시(MUST) 통계 요청이 발급된 RTCPeerConnection 객체가 전송하거나 수신하는 MediaStreamTrack이어야 한다. 호출하는 웹 애플리케이션은 선택자(selector)를 getStats() 메소드에 제공하고 브라우저는 통계 선택 알고리즘에 따라 선택자(selector)와 관련된 일련의 통계를 내보낸다. 이 알고리즘은 선택자(selector)의 송신자(sender)또는 수신자(receiver)를 사용한다는 점에 유의해야 한다.

통계 객체에 반환되는 통계는 반복 쿼리들이 RTCStats id라는 딕셔너리 멤버에 의해 연결할 수 있는 방식으로 설계된다. 따라서 웹 애플리케이션은 해당 기간의 시작과 끝에서 측정을 요청함으로써 주어진 기간에 걸쳐 측정을 할 수 있다.

몇 가지 예외를 제외하고, 모니터링 객체는 한 번 생성되면, 관련 RTCPeerConnection 기간 동안 존재한다. 이렇게 하면 관련 피어 연결이 닫힌 이후에도 getStats()의 결과에서 해당 통계들을 사용할 수 있게 된다.

일부 모니터링 객체만 수명이 짧다. 이러한 객체의 통계는 이후 getStats() 결과에서 더 이상 사용할 수 없다. [WEBRTC-STATS]에서의 객체 설명은 이러한 모니터링되는 객체가 삭제되는 시기를 설명한다.

(2) RTCPeerConnection 인터페이스 확장

통계 API는 아래에 설명된 RTCPeerConnection 인터페이스를 확장한다.

WebIDL

```
partial interface RTCPeerConnection {
 Promise<RTCStatsReport> getStats(optional MediaStreamTrack? selector = null);
};
```

메소드

getStats

지정된 셀렉터(selector)에 대한 통계를 수집하고 결과를 비동기식으로 보고한다.

getStats() 메소드가 호출되면 사용자 에이전트는 반드시(MUST) 다음 단계를 실행해야 한다:

1. selectorArg를 메소드의 첫 번째 인자(argument)로 둔다.
2. 연결을 메소드가 호출된 RTCPeerConnection 객체로 둔다.
3. 만약 selectorArg가 null이라면, 선택자(selector)를 null로 유지한다.
4. 만약 selectorArg가 MediaStreamTrack인 경우, 연결 시 선택자(selector)를 selector Arg와 일치하는 트랙 속성, RTCRtpSender 또는 RTCRtpReceiver로 설정한다. 이러한 송신자(sender) 또는 수신자(receiver)가 존재하지 않거나 둘 이상의 송신자(sender) 또는 수신자(receiver)가 이 기준에 적합한 경우 새로 생성된 InvalidAccessError로 거부된 Promise를 리턴한다.
5. p를 새로운 promise로 지정한다.
6. 다음 단계를 병렬로 실행한다:
   1) 통계 선택 알고리즘에 따라 선택자(selector)로 표시된 통계를 수집한다.
   2) 수집된 통계를 포함하는 결과 RTCStatsReport 객체로 p를 resolve한다.
7. p를 리턴.

(3) RTCStatsReport 객체

getStats() 메소드는 RTCStatsReport 객체의 형태로 성공적인 결과를 전달한다. RTCStats Report 객체는 검사된 객체(RTCStats 인스턴스의 id 속성)와 해당 RTCStats에서 파생된 딕셔너리를 식별하는 문자열 사이의 맵(map)이다.

RTCStatsReport는 여러 개의 RTCStats에서 파생된 딕셔너리로 구성될 수 있으며, 그것의 구현은 선택자(selector)와 관련이 있다고 생각하는 하나의 기본 객체에 대한 각각의 보고 통계로 구성될 수 있다. 특정 타입의 모든 통계를 합산하여 선택자(selector)의 총계를 달성한다; 예를 들어, RTCRtpSender가 네트워크를 통해 트랙을 전달하기 위해 여러 SSRC를 사용하는 경우, RTCStatsReport는 SSRC당 하나의 RTCStats에서 파생된 딕셔너리(ssrc Stats 속성 값으로 구분할 수 있음)을 포함할 수 있다.

WebIDL

```
[Exposed=Window]
interface RTCStatsReport {
 readonly maplike<DOMString, object>;
};
```

이 통계 리포트가 구성된 RTCStats에서 가져온 다양한 딕셔너리를 검색하려면 이 딕셔너리를 사용하십시오. 지원되는 속성 이름 집합 [WEBIDL]은 이 통계 리포트에 대해 생성된 모든 RTCStats에서 파생된 딕셔너리의 id로 정의된다.

(4) RTCStats 딕셔너리

RTCStats 딕셔너리는 모니터링되는 특정 개체를 검사하여 생성한 통계 개체를 나타낸다. RTCStats 딕셔너리는 타임스탬프 및 타입과 같은 기본 속성 집합으로 지정하는 기본 타입이다. RTCStats 딕셔너리를 확장하여 특정 통계를 추가한다.

통계 이름은 표준화되지만, 주어진 구현은 실험 값이나 웹 애플리케이션에 아직 알려지지 않은 값을 사용할 수 있다는 점에 유의하십시오. 따라서 애플리케이션은 반드시 알 수 없는 통계를 처리할 수 있도록 준비해야 한다.

계산에서 합리적인 값을 산출하기 위해 통계는 서로 동기화되어야 한다. 예를 들어, bytesSent와 packetsSent가 모두 보고된 경우, 둘 다 동일한 간격으로 보고되어야 "평균 패킷 크기"가 "바이트/패킷"으로 계산될 수 있다. 간격이 서로 다르면 오류가 발생한다. 따라서 구현은 반드시 RTCStats에서 파생된 딕셔너리의 모든 통계에 대해 동기화된 값을 반환해야 한다.

WebIDL

```
dictionary RTCStats {
 required DOMHighResTimeStamp timestamp;
 required RTCStatsType type;
 required DOMString id;
};
```

RTCStats 딕셔너리 멤버

timestamp : DOMHighResTimeStamp 유형

이 객체와 연결된 timestamp 타입의 DOMHighResTimeStamp. 시간은 UNIX 에포크(epoch)에 상대적이다. 원격 소스(예: RTCP 패킷 수신)에서 나온 통계에 대한 타임스탬프는 정보가 로컬 엔드포인트에 도착한 시간을 나타낸다. 원격 타임스탬프는 이와 같이 해당하는 경우 RTCStats에서 파생된 딕셔너리의 추가 필드에서 찾을 수 있다.

type : RTCStatsType 유형

이 객체의 타입.

type 속성은 반드시(MUST) RTCStats 딕셔너리가 나타내는 가장 구체적인 형식의 이름으로 초기화되어야 한다.

id : DOMString 유형

RTCStats 객체를 생성하기 위해 검사한 객체와 관련된 고유 id. 서로 다른 두 개의 RTCStats Report 객체에서 추출된 두 개의 RTCStats 객체는 동일한 기본 객체를 검사하여 생성된 경우 반드시 동일한 id를 가져야한다.
통계 id는 애플리케이션에서 반드시 예측할 수 없어야 한다. 이것은 특정 사용자 에이전트의 id 생성 방식에 따라 애플리케이션이 해당 특정 통계 객체의 ID를 이미 읽지 않은 경우 id로 통계 객체를 가져올 수 없도록 하기 때문이다.
사용자 에이전트는 위의 요구 사항을 만족하는 한 id의 형식을 자유롭게 선택할 수 있다.

참고

사용자 에이전트는 피어 연결에 고유한 NaCl(Networking and Cryptography Library, pronounced salt)을 사용하는 한 생성된 예측 가능한 문자열을 해시 함수를 사용하여 예측할 수 없는 문자열로 변환할 수 있다. 이를 통해 구현은 내부적으로는 예측 가능한 id를 가질 수 있으며, 이는 통계 객체가 getStats() 호출에 걸쳐 안정적인 id를 가지고 있음을 보다 쉽게 보장할 수 있다.

RTCStatsType에 대한 유효한 값들의 집합과 RTCStats에서 파생된 딕셔너리들이 [WEBRTC-STATS]에 문서화되어 있다.

(5) 통계 선택 알고리즘

통계 선택 알고리즘은 다음과 같다:

1. result를 비어있는 be an empty RTCStatsReport로 둔다.
2. 만약 선택자(selector)가 null인 경우, 전체 연결에 대한 통계를 수집하여 결과에 추가하고, 결과를 리턴한 후, 이 단계를 중단한다.
3. 만약 선택자(selector)가 RTCRtpSender인 경우, 통계를 수집하고 다음 객체를 추가한다:
   - 선택자(selector)에 의하여 전송되는 RTP 스트림을 나타내는 모든 RTCOutbound RtpStreamStats 객체.
   - 추가된 RTCOutboundRtpStreamStats 객체에서 직접 또는 간접적으로 참조하는 모든 통계 객체.
4. 만약 선택자(selector)가 RTCRtpReceiver인 경우, 통계를 수집하고 다음 객체를 추가한다:
   - 선택자(selector)에 의하여 전송되는 RTP 스트림을 나타내는 모든 RTCInbound RtpStreamStats 객체.
   - 추가된 RTCInboundRtpStreamStats 객체에서 직접 또는 간접적으로 참조하는 모든 통계 객체.
5. result를 반환한다.

(6) 통계 구현에서의 필수 사항

[WEBRTC-STATS]에 열거된 통계는 광범위한 사용 사례를 다루기 위한 것이다. 이들 모두가 모든 WebRTC 구현에 의해 구현되어야 하는 것은 아니다.

구현은 RTCPeerConnection에 해당 객체가 있을 때 RTCStats 딕셔너리에 정의된 일반 필드 외에 해당 객체에 유효한 필드가 등록되는 경우, 다음 타입의 통계를 생성하도록 반드시(MUST) 지원해야 한다.

RTCStatsType	Dictionary	Fields
"codec"	RTCCodecStats	payloadType, codecType, mimeType, clockRate, channels, sdpFmtpLine
"inbound-rtp"	RTCRtpStreamStats	ssrc, kind, transportId, codecId
	RTCReceivedRtpStreamStats	packetsReceived, packetsLost, jitter, packetsDiscarded, framesDropped
	RTCInboundRtpStreamStats	receiverId, remoteId, framesDecoded, nackCount, framesReceived, bytesReceived, totalAudioEnergy, totalSamplesDuration
"outbound-rtp"	RTCRtpStreamStats	ssrc, kind, transportId, codecId
	RTCSentRtpStreamStats	packetsSent, bytesSent
	RTCOutboundRtpStreamStats	senderId, remoteId, framesEncoded, nackCount, framesSent
"remote-inbound-rtp"	RTCRtpStreamStats	ssrc, kind, transportId, codecId
	RTCReceivedRtpStreamStats	packetsReceived, packetsLost, jitter, packetsDiscarded, framesDropped
	RTCRemoteInboundRtpStreamStats	localId, roundTripTime
"remote-outbound-rtp"	RTCRtpStreamStats	ssrc, kind, transportId, codecId
	RTCSentRtpStreamStats	packetsSent, bytesSent
	RTCRemoteOutboundRtpStreamStats	localId, remoteTimestamp
"media-source"	RTCMediaSourceStats	trackIdentifier, kind
	RTCAudioSourceStats	totalAudioEnergy, totalSamplesDuration (송신자에게 부착된 오디오 트랙에 대해)
	RTCVideoSourceStats	width, height, framesPerSecond (송신자에게 부착된 비디오 트랙에 대해)
"peer-connection"	RTCPeerConnectionStats	dataChannelsOpened, dataChannelsClosed
"data-channel"	RTCDataChannelStats	label, protocol, dataChannelIdentifier, state, messagesSent, bytesSent, messagesReceived, bytesReceived
"sender"	RTCMediaHandlerStats	trackIdentifier
"receiver"		

RTCStatsType	Dictionary	Fields
"transport"	RTCTransportStats	bytesSent, bytesReceived, selectedCandidatePairId, localCertificateId, remoteCertificateId
"candidate-pair"	RTCIceCandidatePairStats	transportId, localCandidateId, remoteCandidateId, state, nominated, bytesSent, bytesReceived, totalRoundTripTime, currentRoundTripTime
"local-candidate"	RTCIceCandidateStats	address, port, protocol, candidateType, url
"remote-candidate"		
"certificate"	RTCCertificateStats	fingerprint, fingerprintAlgorithm, base64Certificate, issuerCertificateId

구현은 [WEBRTC-STATS]에 정의된 통계들과 다른 통계를 생성할 수 있고, 문서화되지 않은 통계를 생성할 수 있다.

(7) GetStats 예제

사용자가 좋지 않은 음질을 경험하고 있는 경우 애플리케이션에서 그 원인이 패킷 손실인지의 여부를 확인하려 하는 경우를 고려해야 한다. 다음과 같은 예시 코드가 사용될 수 있다:

EXAMPLE 8

```
async function gatherStats(pc) {
 try {
 const [sender] = pc.getSenders();
 const baselineReport = await sender.getStats();
 await new Promise(resolve => setTimeout(resolve, aBit)); // 잠시 기다린다
 const currentReport = await sender.getStats();

 // 기준과 현재 리포트의 요소를 비교
 for (const now of currentReport.values()) {
  if (now.type != 'outbound-rtp') continue;

  // 기준 리포트에서 현재 통계 가져오기
  const base = baselineReport.get(now.id);
  if (!base) continue;

  const remoteNow = currentReport.get(now.remoteId);
  const remoteBase = baselineReport.get(base.remoteId);

  const packetsSent = now.packetsSent - base.packetsSent;
  const packetsReceived = remoteNow.packetsReceived -
        remoteBase.packetsReceived;

  const fractionLost = (packetsSent - packetsReceived) / packetsSent;
  if (fractionLost > 0.3) {
  // 만약 fractionLost > 0.3 이라면, 우리는 아마도 문제의 원인을 찾을수 있을 것이다
  }
 }
 } catch (err) {
 console.error(err);
 }
}
```

9) 네트워크 사용을 위한 미디어 스트림 API 확장

(1) 소개

[GETUSERMEDIA] 규격에 정의된 MediaStreamTrack 인터페이스는 일반적으로 오디오 또는 비디오의 데이터 스트림을 나타낸다. 하나 이상의 MediaStreamTrack이 MediaStream에서 수집 될 수 있다(엄격히 말해서 [GETUSERMEDIA]에 정의된 MediaStream에는 0개 이상의 MediaStreamTrack 객체가 포함될 수 있음).

MediaStreamTrack은 원격 피어(예 : 로컬 카메라뿐만 아니라)에서 오거나 원격 피어로 전송되는 미디어 흐름을 나타내도록 확장 될 수 있다. MediaStreamTrack 객체에서 이 기능을 활성화하는 데 필요한 확장은 이 절에서 설명한다. 미디어가 피어에게 전송되는 방식은 [RTCWEB-RTP], [RTCWEB-AUDIO], [RTCWEB-TRANSPORT]에 설명되어 있다.

다른 피어에게 전송된 MediaStreamTrack은 수신자에게 하나의 MediaStreamTrack으로 표시된다. 피어는 이 규격을 지원하는 사용자 에이전트로 정의된다. 또한 송신 측 애플리케이션은 MediaStreamTrack이 구성원인 MediaStream 객체를 나타낼 수 있다. 수신자 측의 해당 MediaStream 객체가 생성되고(아직 없는 경우) 그에 따라 채워진다.

이 문서의 앞부분에서도 설명했듯이 RTCRtpSender 및 RTCRtpReceiver 객체는 애플리케이션에서 MediaStreamTrack의 전송 및 수신을 보다 세밀하게 제어하는 데 사용할 수 있다.

채널은 미디어 캡처 및 스트림 규격에서 고려되는 가장 작은 단위이다. 채널은 예를 들어 RTP 페이로드 유형과 같이 전송을 위해 함께 인코딩된다. 코덱이 공동으로 인코딩해야하는 모든 채널은 동일한 MediaStreamTrack에 있어야 하며 (MUST), 코덱은 트랙의 모든 채널을 인코딩하거나 삭제할 수 있어야 한다(SHOULD).

주어진 MediaStreamTrack에 대한 입력 및 출력의 개념은 네트워크를 통해 전송되는 MediaStreamTrack 객체의 경우에도 적용된다. RTCPeerConnection 객체에 의해 생성된 MediaStreamTrack(이 문서의 앞부분에서 설명한 대로)은 원격 피어로부터 수신된 데이터를 입력으로 받는다. 마찬가지로 로컬 소스(예 : [GETUSERMEDIA]를 통한 카메라)의 MediaStreamTrack에는 객체가 RTCPeerConnection 객체와 함께 사용되는 경우 원격 피어로 전송되는 내용을 나타내는 출력이 있다.

[GETUSERMEDIA]에 설명된 대로 MediaStream 및 MediaStreamTrack 객체를 복제하는 개념도 여기에 적용된다. 예를 들어 이 기능은 화상 회의 시나리오에서 사용자의 카메라와 마이크의 로컬 비디오를 로컬 모니터에 표시하는 동시에 원격 피어로만 오디오를 전송하는 데 사용할 수 있다(예 : " 비디오 음소거 "기능). 다른 MediaStreamTrack 객체를 새로운 MediaStream

객체로 결합하는 것은 특정 상황에서 유용하다.

참고

이 문서에서는 RTCPeerConnection과 함께 사용할 때 관련된 다음 객체부분만 명시한다. MediaStream 및 MediaStreamTrack 사용에 대한 일반적인 정보는 [GETUSERMEDIA] 문서에서 객체의 원래 정의를 참조.

(2) MediaStream

① id

MediaStream에 명시된 id 속성은 이 스트림에 고유한 ID를 반환하므로 RTCPeerConnection API의 원격지에서 스트림을 인식 할 수 있다.

원격 피어에서 얻은 스트림을 기술하기 위해 MediaStream이 생성되면 원격 소스에서 제공하는 정보에서 id 속성이 초기화된다.

참고

MediaStream 객체의 id는 스트림 출처에 고유하지만 중복으로 끝날 수 없다는 의미는 아니다. 예를 들어 로컬에서 생성된 스트림의 트랙은 RTCPeerConnection을 사용하여 한 사용자 에이전트에서 원격 피어로 전송된 다음 동일한 방식으로 원래 사용자 에이전트로 다시 전송될 수 있으며, 이 경우 원래 사용자 에이전트는 동일한 id (로컬에서 생성된 id와 원격 피어에서 받은 id)를 가진 여러 스트림을 가질 수 있다.

(3) MediaStreamTrack

로컬이 아닌 미디어 소스의 경우(RTCRtpReceiver와 연결된 각 MediaStreamTrack의 경우 같은 RTP 소스)에서 MediaStreamTrack 객체의 MediaStream 참조는 항상 강력하다.

RTCRtpReceiver가 해당 MediaStreamTrack이 음소거되었지만 종료되지 않은 RTP 소스에서 데이터를 수신할 때마다, RTCRtpReceiver가 구성원 인 RTCRtpTransceiver 객체의 [[Receptive]] 슬롯이 true일 때마다, 해당 MediaStreamTrack의 음소거 상태를 false로 설정하는 작업을 반드시(MUST) 큐에 넣어야 한다.

RTCRtpReceiver가 수신한 RTP 소스 미디어 스트림에 대한 SSRC 중 하나가 BYE 수신 또는 시간 제한을 통해 제거되면 해당 MediaStreamTrack의 음소거 상태를 true로 설정하는 작업을 반드시(MUST) 큐에 넣어야 한다. setRemoteDescription은 트랙의 음소거 상태를 true 값으로 설정할 수도 있다.

트랙 추가, 트랙 제거 및 트랙의 음소거 상태 설정 절차는 [GETUSERMEDIA]에 명시되어 있다.

RTCRtpReceiver 수신기에 의해 생성된 MediaStreamTrack 트랙이 [GETUSERMEDIA]를 종료하면(예 : receiver.track.stop 호출을 통해), 사용자 에이전트는 예를 들어 수신기의 디코더를 종료함으로 수신 스트림에 할당된 리소스를 해제하도록 선택할 수 있다(MAY).

① MediaTrackSupportedConstraints, MediaTrackCapabilities, MediaTrackConstraints, MediaTrackSettings

MediaTrackConstraints (MediaStreamTrack.getConstraints (), MediaStreamTrack.applyConstraints ()) 및 MediaTrackSettings (MediaStreamTrack.getSettings ())를 포함한 제약 조건 및 제약 가능한 속성의 개념은 [GETUSERMEDIA]에 설명되어 있다. 그러나 피어 연결에서 가져온 트랙의 제한 가능한 속성은 getUserMedia ()에서 가져온 속성과 다르다; 원격 소스에서 가져온 MediaStreamTrack에 적용 할 수 있는 제약 및 설정이 여기에 정의되어 있다. 원격 트랙의 설정은 수신된 최신 프레임을 나타낸다.

MediaStreamTrack.getCapabilities ()는 반드시(MUST) 항상 빈 집합을 반환해야하며, MediaStreamTrack.applyConstraints ()는 여기에 정의된 제약 조건에 대해 반드시(MUST) 원격 트랙에서 항상 OverconstrainedError로 거부해야한다.

다음 제한 가능한 속성은 원격 소스에서 가져온 비디오 MediaStreamTrack에 적용하기 위해 정의된다:

속성 이름	값	참고
width	ConstrainULong	설정으로서, 이것은 가장 최근에 받은 프레임의 너비(픽셀)이다.
height	ConstrainULong	설정으로서, 이것은 가장 최근에 받은 프레임의 높이(픽셀)이다.
frameRate	ConstrainDouble	설정으로서, 이것은 최근에 수신한 프레임을 바탕으로 한 프레임률의 추정치다.
aspectRatio	ConstrainDouble	설정으로서, 이것은 가장 최근의 프레임의 가로 세로 비율이다; 이것은 픽셀의 폭을 픽셀 단위로 나눈 폭이다. 10번째 소수점 자리까지 반올림한 것이다.

이 문서는 원격 소스에서 가져온 오디오 MediaStreamTrack에 적용할 제한 가능한 속성을 정의하지 않는다.

10) 예제와 Call 흐름

이 섹션은 비표준이다.

(1) 간단한 피어투피어(호출) 예제

두 피어가 서로 연결을 설정하기로 결정하면, 둘 다 이런 단계를 거친다. STUN / TURN 서버 구성은 공용 IP 주소 등을 가져 오거나 NAT 통과를 설정하는 데 사용할 수 있는 서버를 설명한다. 또한 그들은 처음에 통신할 것이라는 것을 설정하는 데 사용한 것과 동일한 대역 외 메커니즘을 사용하여 신호 채널에 대한 데이터를 서로에게 전송해야한다.

Example 9

```
const signaling = new SignalingChannel(); // handles JSON.stringify/parse
const constraints = {audio: true, video: true};
const configuration = {iceServers: [{urls: 'stun:stun.example.org'}]};
const pc = new RTCPeerConnection(configuration);

// 다른 피어에게 ice 후보를 보낸다
pc.onicecandidate = ({candidate}) => signaling.send({candidate});

// "negotiationneeded" 이벤트를 offer 생성하도록 한다(trigger)
pc.onnegotiationneeded = async () => {
 try {
 await pc.setLocalDescription();
 // 다른 피어에게 offer를 보낸다
 signaling.send({description: pc.localDescription});
 } catch (err) {
 console.error(err);
```

```
 }
};

pc.ontrack = ({track, streams}) => {
 // 원격 트랙용 미디어가 도착하면, 원격 비디오 요소에 표시
 track.onunmute = () => {
 // 만약 srcObject 가 설정되어 있으면, 다시 설정하지 않는다
 if (remoteView.srcObject) return;
 remoteView.srcObject = streams[0];
 };
};

// call start() - 초기화를 위한
function start() {
 addCameraMic();
}

// 연결을 위한 카메라 마이크 추가
async function addCameraMic() {
 try {
 // 로컬 스트림을 얻고, 셀프뷰에 보여주고 전송을 위해 추가하기
 const stream = await navigator.mediaDevices.getUserMedia(constraints);
 for (const track of stream.getTracks()) {
  pc.addTrack(track, stream);
 }
 selfView.srcObject = stream;
 } catch (err) {
 console.error(err);
 }
}
```

```
signaling.onmessage = async ({data: {description, candidate}}) => {
 try {
 if (description) {
  await pc.setRemoteDescription(description);
  // 만약 offer를 받았다면, answer로 답해야 함
  if (description.type == 'offer') {
  if (!selfView.srcObject) {
   // 권한에 대한 협상 차단(제품 코드에서는 권장되지 않음)
   await addCameraMic();
  }
  await pc.setLocalDescription();
  signaling.send({description: pc.localDescription});
  }
 } else if (candidate) {
  await pc.addIceCandidate(candidate);
 }
 } catch (err) {
 console.error(err);
 }
};
```

(2) 워밍업을 사용한 고급 피어투피어 예제

두 피어가 서로 연결을 설정하고 ICE, DTLS 및 미디어 연결을 "워밍업" 하여 즉시 미디어를 보내고 받을 준비가 되도록 결정하면 둘 다 이런 단계들을 거친다.

Example 10

```
const signaling = new SignalingChannel(); // 핸들 JSON.stringify/parse
const constraints = {audio: true, video: true};
const configuration = {iceServers: [{urls: 'stun:stun.example.org'}]};
let pc;
let audio;
let video;
let started = false;

// Call warmup() - 미디어가 준비되기 전에, ICE, DTLS와 media 를 warmup
async function warmup(isAnswerer) {
 pc = new RTCPeerConnection(configuration);
 if (!isAnswerer) {
 audio = pc.addTransceiver('audio');
 video = pc.addTransceiver('video');
 }

 // ice 후보를 다른 피어에게 보냄
 pc.onicecandidate = ({candidate}) => signaling.send({candidate});

 // "negotiationneeded" 이벤트를 offer 생성하도록 한다(trigger)
 pc.onnegotiationneeded = async () => {
 try {
  await pc.setLocalDescription();
  // offer를 다른 피어에게 보냄
  signaling.send({description: pc.localDescription});
```

```
} catch (err) {
 console.error(err);
}
};

pc.ontrack = async ({track, transceiver}) => {
try {
 // 원격 트랙용 미디어가 도착하면, 비디오 엘리먼트에서 보여줌
 event.track.onunmute = () => {
 // 만약 srcObject가 이미 설정되어 있다면, 다시 설정하지 말 것
 if (!remoteView.srcObject) {
  remoteView.srcObject = new MediaStream();
 }
 remoteView.srcObject.addTrack(track);
 }

 if (isAnswerer) {
 if (track.kind == 'audio') {
  audio = transceiver;
 } else if (track.kind == 'video') {
  video = transceiver;
 }
 if (started) await addCameraMicWarmedUp();
 }
} catch (err) {
 console.error(err);
}
};

try {
```

```
 // 로컬 스트림을 얻고, 셀프뷰에 보여주고 전송을 위해 추가하기
 selfView.srcObject = await navigator.mediaDevices.getUserMedia(constraints);
 if (started) await addCameraMicWarmedUp();
 } catch (err) {
 console.error(err);
 }
}

// 워밍업 후 start() 를 호출하여 양단간에 미디어 전송을 시작
function start() {
 signaling.send({start: true});
 signaling.onmessage({data: {start: true}});
}

// add camera and microphone to already warmed-up connection
async function addCameraMicWarmedUp() {
 const stream = selfView.srcObject;
 if (audio && video && stream) {
 await Promise.all([
  audio.sender.replaceTrack(stream.getAudioTracks()[0]),
  video.sender.replaceTrack(stream.getVideoTracks()[0]),
 ]);
 }
}

signaling.onmessage = async ({data: {start, description, candidate}}) => {
 if (!pc) warmup(true);

 try {
 if (start) {
```

```
    started = true;
    await addCameraMicWarmedUp();
  } else if (description) {
    await pc.setRemoteDescription(description);
    // if we got an offer, we need to reply with an answer
    if (description.type == 'offer') {
    await pc.setLocalDescription();
    signaling.send({description: pc.localDescription});
    }
  } else {
    await pc.addIceCandidate(candidate);
  }
  } catch (err) {
  console.error(err);
  }
};
```

(3) Simulcast Example

클라이언트가 여러 RTP 인코딩(시뮬 캐스트)을 서버로 보내려고 한다.

Example 11

```
const signaling = new SignalingChannel(); // handles JSON.stringify/parse
const constraints = {audio: true, video: true};
const configuration = {'iceServers': [{'urls': 'stun:stun.example.org'}]};
let pc;

// call start() to initiate
async function start() {
 pc = new RTCPeerConnection(configuration);

 // let the "negotiationneeded" event trigger offer generation
 pc.onnegotiationneeded = async () => {
 try {
  await pc.setLocalDescription();
  // send the offer to the other peer
  signaling.send({description: pc.localDescription});
 } catch (err) {
  console.error(err);
 }
 };

 try {
 // get a local stream, show it in a self-view and add it to be sent
 const stream = await navigator.mediaDevices.getUserMedia(constraints);
 selfView.srcObject = stream;
 pc.addTransceiver(stream.getAudioTracks()[0], {direction: 'sendonly'});
 pc.addTransceiver(stream.getVideoTracks()[0], {
```

```
    direction: 'sendonly',
    sendEncodings: [
    {rid: 'q', scaleResolutionDownBy: 4.0}
    {rid: 'h', scaleResolutionDownBy: 2.0},
    {rid: 'f'},
    ]
  });
  } catch (err) {
  console.error(err);
  }
}

signaling.onmessage = async ({data: {description, candidate}}) => {
  try {
  if (description) {
    await pc.setRemoteDescription(description);
    // if we got an offer, we need to reply with an answer
    if (description.type == 'offer') {
    await pc.setLocalDescription();
    signaling.send({description: pc.localDescription});
    }
  } else if (candidate) {
    await pc.addIceCandidate(candidate);
  }
  } catch (err) {
  console.error(err);
  }
};
```

(4) Peer-to-peer Data Example

이 예제는 RTCDataChannel 객체를 생성하고 채널을 다른 피어에 연결하는 데 필요한 제안/응답 교환을 수행하는 방법을 보여준다. RTCDataChannel은 사용자 입력으로 입력필드를 사용하는 간단한 채팅 응용 프로그램의 컨텍스트로 사용된다.

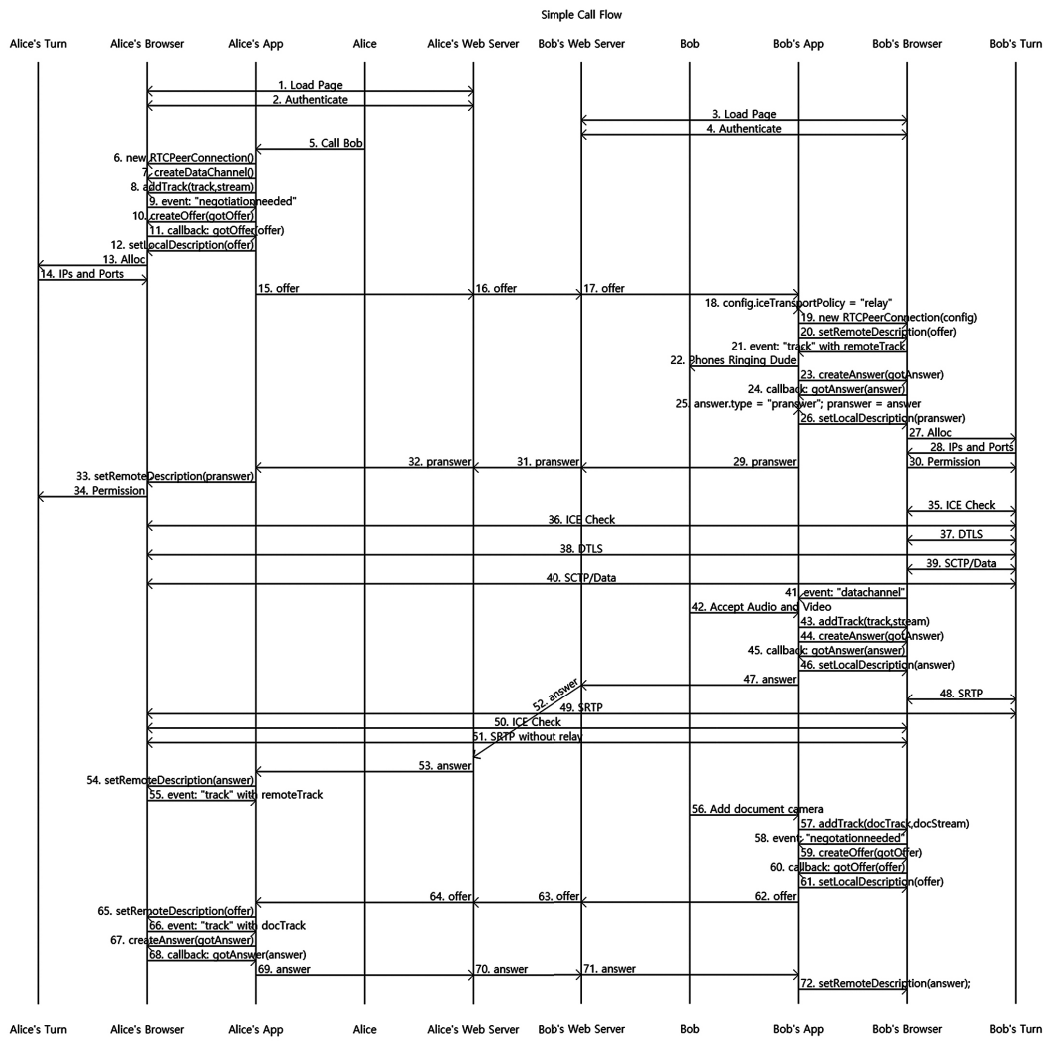
Example 12

```
const signaling = new SignalingChannel(); // handles JSON.stringify/parse
const configuration = {iceServers: [{urls: 'stun:stun.example.org'}]};
let pc, channel;

// call start() to initiate
function start() {
 pc = new RTCPeerConnection(configuration);

 // send any ice candidates to the other peer
 pc.onicecandidate = ({candidate}) => signaling.send({candidate});

 // let the "negotiationneeded" event trigger offer generation
 pc.onnegotiationneeded = async () => {
 try {
  await pc.setLocalDescription();
  // send the offer to the other peer
  signaling.send({description: pc.localDescription});
 } catch (err) {
  console.error(err);
 }
 };

 // create data channel and setup chat using "negotiated" pattern
 channel = pc.createDataChannel('chat', {negotiated: true, id: 0});
```

```
  channel.onopen = () => input.disabled = false;
  channel.onmessage = ({data}) => showChatMessage(data);

  input.onkeypress = ({keyCode}) => {
  // only send when user presses enter
  if (keyCode != 13) return;
  channel.send(input.value);
  }
}

signaling.onmessage = async ({data: {description, candidate}}) => {
  if (!pc) start(false);

  try {
  if (description) {
    await pc.setRemoteDescription(description);
    // if we got an offer, we need to reply with an answer
    if (description.type == 'offer') {
    await pc.setLocalDescription();
    signaling.send({description: pc.localDescription});
    }
  } else if (candidate) {
    await pc.addIceCandidate(candidate);
  }
  } catch (err) {
  console.error(err);
  }
};
```

(5) Call Flow Browser to Browser

이것은 두 브라우저 간의 가능한 호출 흐름의 예를 보여준다. 이것은 로컬 미디어 또는 실행되는 모든 콜백에 액세스하는 절차를 보여주지 않고 대신 주요 이벤트와 메시지만 표시하도록 줄이도록 한 것이다.



(6) DTMF Example

예제는 송신자가 RTCRtpSender라고 가정한 것이다.

DTMF 신호 “1234”를 tone당 500밀리초 간격으로 보낸 것

Example 13

```
if (sender.dtmf.canInsertDTMF) {
 const duration = 500;
 sender.dtmf.insertDTMF('1234', duration);
} else {
 console.log('DTMF function not available');
}
```

DTMF 신호 "123"을 보내고 "2"를 보낸 후 중단한다.

Example 14

```
async function sendDTMF() {
 if (sender.dtmf.canInsertDTMF) {
 sender.dtmf.insertDTMF('123');
 await new Promise(r => sender.dtmf.ontonechange = e => e.tone == '2' && r());
 // empty the buffer to not play any tone after "2"
 sender.dtmf.insertDTMF('');
 } else {
 console.log('DTMF function not available');
 }
}
```

DTMF 신호 "1234"를 전송하고, 활성화 키를 켠다.

```
lightKey(key)
```

톤이 재생되는 동안,

```
lightKey("")
```

(모든 키가 어두워 진다고 가정):

Example 15

```
const wait = ms => new Promise(resolve => setTimeout(resolve, ms));

if (sender.dtmf.canInsertDTMF) {
 const duration = 500; // ms
 sender.dtmf.insertDTMF(sender.dtmf.toneBuffer + '1234', duration);
 sender.dtmf.ontonechange = async ({tone}) => {
 if (!tone) return;
 lightKey(tone); // light up the key when playout starts
 await wait(duration);
 lightKey(''); // turn off the light after tone duration
 };
} else {
 console.log('DTMF function not available');
}
```

항상 톤 버퍼에 추가하는 것이 안전하다. 이 예제는 tone 재생 전과 재생 중에 추가하는 것을 보여준다.

Example 16

```
if (sender.dtmf.canInsertDTMF) {
 sender.dtmf.insertDTMF('123');
 // append more tones to the tone buffer before playout has begun
 sender.dtmf.insertDTMF(sender.dtmf.toneBuffer + '456');

 sender.dtmf.ontonechange = ({tone}) => {
 // append more tones when playout has begun
 if (tone != '1') return;
 sender.dtmf.insertDTMF(sender.dtmf.toneBuffer + '789');
 };
} else {
 console.log('DTMF function not available');
}
```

1 초 "1"tone을 보내고, 2 초 "2"tone을 보낸다.

Example 17

```
if (sender.dtmf.canInsertDTMF) {
 sender.dtmf.ontonechange = ({tone}) => {
 if (tone == '1') {
  sender.dtmf.insertDTMF(sender.dtmf.toneBuffer + '2', 2000);
 }
 };
 sender.dtmf.insertDTMF(sender.dtmf.toneBuffer + '1', 1000);
} else {
 console.log('DTMF function not available');
}
```

(7) 완벽한 협상 예 (Perfect Negotiation Exaple)

완벽한 협상은 협상을 투명하게 관리하는 데 권장되는 패턴으로, 애플리케이션의 나머지 부분에서 비대칭형 작업을 끌어낸다. 이 패턴은 응용 프로그램이 눈부심의 위험 없이 동시에 두 피어 연결 개체에서 작동 할 수 있도록 하므로 한 쪽이 항상 제공자가 되는 것보다 장점이 있다("stable" 상태 외부로 들어오는 제안). 나머지 응용 프로그램은 상태 경합 신호에 대한 걱정 없이 모든 수정 방법 및 속성을 사용할 수 있다.

두 피어 사이의 신호 충돌을 해결하는 동작으로 서로 다른 역할을 두 피어에 지정한다.

1. 정중한(polite) 피어는 들어오는 offer와 충돌을 피하기 위해 롤백을 사용한다.
2. 무례한(impolite) 피어는 충돌할 때 들어오는 offer를 무시한다.

함께 교착 상태가 되지 않는 방식으로 나머지 애플리케이션에 대한 신호를 관리한다. 이 예에서는 지정된 역할을 나타내는 polite부울 변수를 가정한다:

Example 18

```
const signaling = new SignalingChannel(); // handles JSON.stringify/parse
const constraints = {audio: true, video: true};
const configuration = {iceServers: [{urls: 'stun:stun.example.org'}]};
const pc = new RTCPeerConnection(configuration);

// call start() anytime on either end to add camera and microphone to connection
async function start() {
 try {
 const stream = await navigator.mediaDevices.getUserMedia(constraints);
 for (const track of stream.getTracks()) {
  pc.addTrack(track, stream);
 }
 selfView.srcObject = stream;
 } catch (err) {
 console.error(err);
 }
}
```

```
pc.ontrack = ({track, streams}) => {
 // once media for a remote track arrives, show it in the remote video element
 track.onunmute = () => {
 // don't set srcObject again if it is already set.
 if (remoteView.srcObject) return;
 remoteView.srcObject = streams[0];
 };
};

// - The perfect negotiation logic, separated from the rest of the application
---

// keep track of some negotiation state to prevent races and errors
let makingOffer = false;
let ignoreOffer = false;
let isSettingRemoteAnswerPending = false;

// send any ice candidates to the other peer
pc.onicecandidate = ({candidate}) => signaling.send({candidate});

// let the "negotiationneeded" event trigger offer generation
pc.onnegotiationneeded = async () => {
 try {
 makingOffer = true;
 await pc.setLocalDescription();
 signaling.send({description: pc.localDescription});
 } catch (err) {
  console.error(err);
 } finally {
 makingOffer = false;
```

```
  }
};

signaling.onmessage = async ({data: {description, candidate}}) => {
  try {
  if (description) {
    // An offer may come in while we are busy processing SRD(answer).
    // In this case, we will be in "stable" by the time the offer is processed
    // so it is safe to chain it on our Operations Chain now.
    const readyForOffer =
     !makingOffer &&
     (pc.signalingState == "stable" || isSettingRemoteAnswerPending);
    const offerCollision = description.type == "offer" && !readyForOffer;

    ignoreOffer = !polite && offerCollision;
    if (ignoreOffer) {
    return;
    }
    isSettingRemoteAnswerPending = description.type == "answer";
    await pc.setRemoteDescription(description); // SRD rolls back as needed
    isSettingRemoteAnswerPending = false;
    if (description.type == "offer") {
    await pc.setLocalDescription();
    signaling.send({description: pc.localDescription});
    }
  } else if (candidate) {
    try {
    await pc.addIceCandidate(candidate);
    } catch (err) {
    if (!ignoreOffer) throw err; // Suppress ignored offer's candidates
    }
```

```
  }
  } catch (err) {
  console.error(err);
  }
}
```

이는 타이밍에 민감하며 setLocalDescription(인수없이) 및 setRemoteDescription(암시적 롤백 포함) 버전을 의도적으로 사용하여 서비스되는 다른 신호 메시지와의 경쟁을 방지한다. ignoreOffer 변수가 필요하다. 왜냐하면 무례한(impolite) 쪽의 RTCPeerConnection 오브젝트가 무시된 제안(offer)에 대해 말한 적이 있다. 따라서 우리는 그러한 제안에 속하는 들어오는 후보자의 오류를 억제해야 한다.

11) 에러 핸들링

일부 작업은 RTCError를 유발하거나 발생시킨다. 이는 추가적인 WebRTC 관련 정보를 전달하는 DOMException의 확장이다.

(1) RTCError Interface

WebIDL

```
[Exposed=Window]
interface RTCError : DOMException {
 constructor(RTCErrorInit init, optional DOMString message = "");
 readonly attribute RTCErrorDetailType errorDetail;
 readonly attribute long? sdpLineNumber;
 readonly attribute long? sctpCauseCode;
 readonly attribute unsigned long? receivedAlert;
 readonly attribute unsigned long? sentAlert;
};
```

생성자

constructor()

다음 단계를 실행한다.

1. init를 생성자의 첫 번째 인수로 둔다.
2. message를 생성자의 두 번째 인수로 둔다.
3. e를 새로운 RTCError 객체로 둔다.
4. message 인자를 message로 설정하고 name 인자를 “OperationError”로 설정해서 e의 DOMException 생성자를 호출한다.

참고

이 name에는 레거시 코드의 매핑이 없으므로 e.code는 0을 반환.

5. e의 모든 RTCError 속성이 있는 경우에는 init의 해당 속성 값으로 설정하고, 그렇지 않으면 null로 설정한다.
6. e를 리턴한다.

속성

errorDetail,RTCErrorDetailType 타입, 읽기전용

발생한 오류 유형에 대한 WebRTC 특정 오류 코드.

sdpLineNumber, long 타입, 읽기전용, null 허용

만약 errorDetail이 "sdp-syntax-error"이면 오류가 감지된 줄 번호.(첫줄 번호는 1을 가진다)

sctpCauseCode, long 타입, 읽기전용, null 허용

만약 errorDetail 이 "sctp-failure"이면 실패한 SCTP 협상의 SCTP 원인 코드.

receivedAlert, unsigned long 타입, 읽기전용, null 허용

만약 errorDetail 이 "dtls-failure" 이고 치명적인 DTLS 경고가 수신 된 경우, 이는 수신된 DTLS 경고 값임.

sentAlert, unsigned long 타입, 읽기전용, null 허용

만약 errorDetail 이 "dtls-failure" 이고 치명적인 DTLS 경고가 전송된 경우, 이는 전송된 DTLS 경고 값임.

(위험한 기능) 이슈 2

RTCError에 정의된 모든 숙성은 구현 부족으로 인해 위험에 표시된다(errorDetail, sdpLineNumber, sctpCauseCode, receivedAlert 및 sentAlert). DOMException에서 상속된 속성은 포함되지 않는다.

RTCErrorInit 사전

WebIDL

```
dictionary RTCErrorInit {
 required RTCErrorDetailType errorDetail;
 long sdpLineNumber;
 long sctpCauseCode;
 unsigned long receivedAlert;
 unsigned long sentAlert;
};
```

errorDetail, sdpLineNumber, sctpCauseCode, receivedAlert와 RTCErrorInit의 sentAlert 멤버는 RTCError와 동일한 속성과 정의를 가지고 있다.

(2) RTCErrorDetailType 열거형

WebIDL

```
enum RTCErrorDetailType {
 "data-channel-failure",
 "dtls-failure",
 "fingerprint-failure",
 "sctp-failure",
 "sdp-syntax-error",
 "hardware-encoder-not-available",
 "hardware-encoder-error"
};
```

열거형	설명
data-channel-failure	데이터 채널 실패
dtls-failure	DTLS 협상이 실패했거나 연결이 치명적인 오류로 종료되었음. 메시지에는 오류의 특성 관련 정보가 포함되어 있음. 치명적인 DTLS 경고가 수신된 경우, receivedAlert 속성은 수신된 DTLS 경고 값으로 설정됨. 치명적인 DTLS 경고가 전송된 경우, sentAlert 속성은 전송된 DTLS 경고 값으로 설정됨.
fingerprint-failure	RTCDtlsTransport의 원격 인증서가 SDP에 제공된 지문과 일치하지 않음. 원격 피어가 제공된 지문에 대해 로컬 인증서를 일치시킬 수 없는 경우 이 오류가 생성되지 않음. 대신 "bad_certificate" (42) DTLS 경고가 원격 피어에서 수신되어 "dtls-failure"가 발생할 수 있음.
sctp-failure	SCTP 협상이 실패했거나 연결이 치명적인 오류로 종료되었음. sctpCauseCode 속성은 SCTP 원인 코드로 설정됨.
sdp-syntax-error	SDP 문법이 유효하지 않음. sdpLineNumber 속성은 문법 오류가 감지된 SDP의 행 번호로 설정됨.
hardware-encoder-not-available	요청한 작업에 필요한 하드웨어 인코더 리소스를 사용할 수 없음.
hardware-encoder-error	하드웨어 인코더가 제공된 매개 변수를 지원하지 않음.

(3) RTCErrorEvent 인터페이스

RTCErrorEvent 인터페이스는 RTCError가 이벤트로 발생하는 경우에 대해 정의되어 있다:

WebIDL

```
[Exposed=Window]
interface RTCErrorEvent : Event {
 constructor(DOMString type, RTCErrorEventInit eventInitDict);
 [SameObject] readonly attribute RTCError error;
};
```

생성자

constructor()

새로운 RTCErrorEvent를 생성.

속성

error, RTCError 타입, 읽기전용

이벤트 트리거한 오류를 설명하는 RTCError.

(4) RTCErrorEventInit 사전

WebIDL

```
dictionary RTCErrorEventInit : EventInit {
 required RTCError error;
};
```

RTCErrorEventInit 사전 멤버

error, RTCError 타입

(있는 경우) 이벤트와 관련된 오류를 설명

12) 이벤트 요약

이 섹션은 비표준이다.

다음 이벤트는 RTCDataChannel 개체에서 발생한다.

이벤트 이름	인터페이스	다음 경우에 ...때 발생함
open	Event	RTCDataChannel객체의 기본 데이터 전송이 확립되어 있다(또는 다시 설정).
message	MessageEvent [html]	메시지가 성공적으로 수신되었다
bufferedamountlow	Event	RTCDataChannel객체의 bufferedAmount 위의 감소 bufferedAmountLowThreshold보다 작거나 그와 동일 bufferedAmountLowThreshold.
error	RTCErrorEvent	데이터 채널에서 오류가 발생했다.
closing	Event	RTCDataChannel은 "개체 전환 closing" 상태
close	Event	RTCDataChannel개체의 기본 데이터 전송이 닫혔다.

다음 이벤트는 RTCPeerConnection 개체에서 발생한다.

이벤트 이름	인터페이스	다음 경우에 ...때 발생함
track	RTCTrackEvent	새로운 수신 미디어가 특정에 대해 협상 RTCRtpReceiver되었으며 해당 수신기가 track연결된 원격에 추가 되었다 MediaStream.
negotiationneeded	Event	브라우저는 세션 협상이 완료되어야 함을 애플리케이션에 알리려고 한다 (예 : createOffer 호출에 이어 setLocalDescription).
signalingstatechange	Event	시그널링 상태 변경되었다. 이 상태 변경은 setLocalDescription 또는 setRemoteDescription호출의 결과이다 .
iceconnectionstatechange	Event	RTCPeerConnection의 ICE 연결 상태가 변경되었다.
icegatheringstatechange	Event	RTCPeerConnection의 ICE 수집 상태가 변경되었다.
icecandidate	RTCPeerConnectionIce Event	RTCIceCandidate스크립트에 새로운 기능이 제공된다.

이벤트 이름	인터페이스	다음 경우에 ...때 발생함
connectionstatechange	Event	RTCPeerConnection. connectionState변경되었다.
icecandidateerror	RTCPeerConnectionIce ErrorEvent	ICE 후보를 수집할 때 실패가 발생했다.
데이터 채널	RTCDataChannelEvent	RTCDataChannel채널을 생성하는 다른 피어에 대한 응답으로 새 스크립트가 스크립트에 전달된다.

다음 이벤트는 RTCDTMFSender 개체에서 발생한다.

이벤트 이름	인터페이스	다음 경우에 ...때 발생함
tonechange	RTCDTMFTone ChangeEvent	RTCDTMFSender 객체는 방금 톤 재생을 시작했거나(tone 속성으로 반환 됨) toneBuffer에서 톤 재생을 막 종료했습니다.(tone 속성에 빈 값으로 반환됨)

다음 이벤트는 RTCIceTransport 개체에서 발생한다.

이벤트 이름	인터페이스	다음 경우에 ...때 발생함
statechange	Event	RTCIceTransport 상태가 변경된다.
gatheringstatechange	Event	RTCIceTransport 수집 상태 변경.
selectedcandidatepairchange	Event	RTCIceTransport 선택된 후보 쌍 변경

다음 이벤트는 RTCDtlsTransport 개체에서 발생한다.

이벤트 이름	인터페이스	다음 경우에 ...때 발생함
statechange	Event	RTCDtlsTransport 상태가 변경된다.
error	RTCErrorEvent	RTCDtlsTransport ("dtls-failure"또는 "fingerprint-failure")에서 오류가 발생했다.

다음 이벤트는 RTCSctpTransport 개체에서 발생한다.

이벤트 이름	인터페이스	다음 경우에 ...때 발생함
statechange	Event	RTCSctpTransport 상태가 변경된다.

13) 개인정보 및 보안 고려사항

이 섹션은 비표준이다.

새로운 동작을 지정하지 않지만 대신 규격의 다른 부분에 이미 있는 정보를 요약한다. WebRTC에서 사용되는 일반적인 API 및 프로토콜 집합에 대한 전반적인 보안 고려 사항은 [RTCWEB-SECURITY-ARCH]에 설명되어 있다.

(1) 동일한 출처 정책에 미치는 영향

이 문서는 브라우저와 다른 브라우저를 포함한 다른 장치 간의 실시간 직접 통신을 설정할 수 있는 기능으로 웹 플랫폼을 확장한다.

데이터와 미디어는 서로 다른 브라우저에서 실행되는 애플리케이션 간에 또는 동일한 브라우저에서 실행 중인 애플리케이션과 브라우저가 아닌 것 간에 공유될 수 있다는 것을 의미한다. 이는 출처가 다른 개체간의 데이터 전송에 대한 웹 모델의 일반적인 장벽을 확장 한 것이다.

WebRTC 규격은 통신을 위한 사용자 프롬프트 또는 크롬 표시기를 제공하지 않는다. 웹 페이지가 미디어에 액세스 할 수 있게 되면 선택한대로 다른 개체와 해당 미디어를 자유롭게 공유 할 수 있다고 가정한다. 따라서 데이터를 보는 WebRTC 데이터 채널의 P2P 교환은 사용자의 명시적 동의나 개입 없이도 발생할 수 있으며, 서버 매개 교환(예 : 웹 소켓을 통해)이 사용자 개입 없이 발생할 수 있는 것과 유사하다.

(2) 공개되는 IP 주소

WebRTC가 없어도 웹 애플리케이션을 제공하는 웹 서버는 애플리케이션이 전달되는 공용 IP 주소를 알고 있다. 통신을 설정하면 브라우저의 네트워크 컨텍스트에 대한 추가 정보가 웹 애플리케이션에 노출되고 WebRTC 사용을 위해 브라우저에서 사용할 수 있는 (비공개) IP 주소 세트가 포함될 수 있다. 이 정보 중 일부는 통신 세션을 설정할 수 있도록 해당 당사자에게 전달되어야 한다.

IP 주소를 공개하면 위치와 연결 수단이 유출될 수 있다; 이것은 민감할 수 있다. 네트워크 환경에 따라 핑거프린트 표면을 늘리고 사용자가 쉽게 지울 수 없는 지속적인 교차 출처 상태를 생성할 수도 있다.

연결은 항상 해당 당사자와의 통신을 위해 제안된 IP 주소를 표시한다. 애플리케이션은 RTCIceTransportPolicy 사전에 의해 노출된 설정을 사용하여 특정 주소를 사용하지 않도록 선택하고

참가자 간의 직접 연결보다는 릴레이(예 : TURN 서버)를 사용하여 이러한 노출을 제한할 수 있다. 일반적으로 TURN 서버의 IP 주소는 민감한 정보가 아니라고 가정한다. 예를 들어 이러한 선택은 사용자가 다른 당사자와 미디어 연결을 시작하는 데 동의했는지 여부에 따라 애플리케이션에서 수행할 수 있다.

애플리케이션 자체에 대한 IP 주소 노출을 완화하려면 사용할 수 있는 IP 주소를 제한해야하며, 이는 엔드 포인트 간의 가장 직접적인 경로에서 통신하는 기능에 영향을 미친다. 브라우저는 사용자가 원하는 보안 상태에 따라 애플리케이션에 사용할 수 있는 IP 주소를 결정하기 위한 적절한 제어를 제공하는 것이 좋다. 노출할 주소의 선택은 로컬 정책에 의해 제어된다(자세한 내용은 [RTCWEB- IP-HANDLING] 참조).

(3) 로컬 네트워크에 미치는 영향

브라우저는 신뢰할 수 있는 네트워크 환경(방화벽 내부)에서 실행되는 활성 플랫폼이므로 브라우저가 로컬 네트워크의 다른 요소에 미칠 수 있는 피해를 제한하는 것이 중요하며 신뢰할 수 없는 참가자에 의한 데이터를 가로 채거나 조작, 수정하지 못하도록 보호하는 것이 중요하다.

완화에는 다음이 포함된다:

- 사용자 에이전트는 항상 ICE를 사용하여 통신하기 위해 해당 사용자 에이전트의 권한을 요청한다. 이렇게 하면 사용자 에이전트가 자격 증명을 공유 한 파트너에게만 보낼 수 있다.
- 항상 사용자 에이전트는 ICE 지속 동의를 사용하여 전송을 계속하기 위해 지속적인 권한을 요청한다. 이를 통해 수신자는 수신 동의를 철회할 수 있다.
- 사용자 에이전트는 항상 강력한 세션 별 키잉(DTLS-SRTP)을 사용하여 데이터를 암호화한다.
- 사용자 에이전트는 항상 혼잡 제어를 사용한다. 이렇게 하면 WebRTC를 사용하여 네트워크를 플러딩할 수 없다.

이러한 조치는 관련 IETF 문서에 명시되어 있다.

(4) 통신 신뢰성

통신이 이루어지고 있다는 사실은 네트워크를 관찰할 수 있는 상대방에게 숨길 수 없으므로 이는 공개 정보로 간주되어야 한다.

통신 인증서는 앞으로의 요구를 예상하여 postMessage()를 사용하여 불투명하게 공유될 수 있다. 사용자 에이전트는 메모리 공격 표면을 줄이기 위해 RTCCertificate 객체에 대한 접근 권한이 있는 프로세스에서 이러한 객체가 핸들을 보유한 개인 키 자료를 분리하는 것이 좋다.

(5) WebRTC로 인해 노출되는 영구적인 정보

위에서 설명한 대로 WebRTC API에 의해 노출 된 IP 주소 목록은 지속적인 교차 출처 상태로 사용될 수 있다.

IP 주소 외에도 WebRTC API는 RTCRtpSender.getCapabilities 및 RTCRtpReceiver.getCapabilities 메소드를 통해 기본 미디어 시스템에 대한 정보를 노출한다. 여기에는 시스템이 생성하고 사용할 수 있는 코덱에 대한 상세하고 정렬된 정보가 포함된다. 해당 정보의 하위 집합은 세션 협상 중에 생성, 노출 및 전송되는 SDP 세션 설명에 표시될 수 있다. 이 정보는 대부분의 경우 시간과 출처에 걸쳐 영구적이며 주어진 장치의 핑거프린트 표면을 증가시킨다.

DTLS 연결을 설정할 때 WebRTC API는 애플리케이션에서 유지할 수 있는 인증서를 생성할 수 있다(예 : IndexedDB). 이러한 인증서는 원본 간에 공유되지 않으며 원본에 대한 영구 스토리지가 지워지면 지워진다.

(6) 원격 엔드 포인트에서 SDP 설정

setRemoteDescription은 예외를 발생시켜 잘못된 SDP를 보호하지만 애플리케이션에서 예상치 못한 SDP에 대해서는 보호하지 않는다. 원격 설명을 설정하면 다른 무엇보다도 중요한 리소스(이미지 버퍼 및 네트워크 포트 포함)가 할당되고 미디어 흐름이 시작될 수 있다(개인 정보 보호 및 대역폭 영향이 있을 수 있음). 악의적인 SDP로부터 보호하지 않는 애플리케이션은 리소스 부족 위험에 노출되어 의도하지 않게 들어오는 미디어를 허용하거나 다른 엔드 포인트가 전송을 협상하지 않는 경우 ontrack과 같은 특정 이벤트가 발생하지 않을 위험이 있다. 애플리케이션은 악의적인 SDP로부터 보호되어야 한다.

14) 접근성 고려사항

이 섹션은 비표준이다.

WebRTC 1.0 규격은 실시간으로 오디오, 비디오와 데이터를 교환을 수립하는 데 필요한 프로토콜(IETF에 정의된)을 제어하기 위한 API를 표현한다.

청각장애인을 위한 통신기기(TTY/TDD- Teleprinter or Teletypewriter/Telecommunications Device for the Deaf)는 청각 장애나 언어장애를 가진 사람들을 전화로 소통할 수 있도록 한다.

[RFC4103]에서 정의한 실시간 텍스트는 RTP에 캡슐화된 T.140을 사용하여, PSAP(Public Safety Access Points)와의 긴급 통신을 포함한, TDD/TTY 장치에서 IP 기반 통신으로 전환이 가능하록 한다.

실시간 텍스트는 거의 실시간으로 보내고 받을 수 있어야 하기 때문에, WebRTC 1.0 데이터 채널 API를 통해 지원하는 것이 가장 좋을 것이다. IETF에서 정의한 대로, 데이터 채널 프로토콜은 신뢰하거나 신뢰할 수 없는 데이터 채널을 모두 지원하는 SCTP/DTLS/UDB 프로토콜 스택을 사용한다. IETF는 SRTP 키 관리와 신뢰할 수 없는 통신에 중점을 둔 RTP 데이터 채널을 위한 제안보다 SCTP/DTLS/UDP를 표준화하기로 했다.

IETF가 WebRTC 프로토콜 모음 중 일부러 RTP 데이터 채널보다 다른 접근을 선택했기 때문에, 이 문서가 발행된 현재 IETF에 정의되거나, 미국 내 규정에는 실시간 텍스트를 직접적으로 지원하는 WebRTC API를 위한 표준화된 방법은 없다. WebRTC 워킹 그룹은 이 영역에 대해서 IETF가 개발하고 있는 프로토콜이 브라우저 API에 직접 적용되기에 적합한지 평가할 것이며, 이 잠재적인 차이에 대한 관련 커뮤니티로부터 의견을 구하고 있다.

IETF MMUSIC(Multiparty Multimedia Session Control) 워킹그룹 내에서는 WebRTC 데이터 채널을 통한 실시간 텍스트를 가능하도록 진행 중이며, SCTP 프로토콜과 RTF 4103 실시간 텍스트 간에 변환이 가능하게 게이트웨이가 배포될 것이다. 이 작업이 완료되면, 게이트웨이나 다른 방법을 통해서, WebRTC user-agents(브라우저 포함)에서 통합적이고 상호 운용성 접근이 가능해질 것으로 기대된다.

이 문서가 발행된 당시에는 WebRTC 클라이언트에서 효율적인 RTT 통신을 가능하게 하는 게이트웨이는 예를 들면 사용자 지정한 WebRTC 데이터 채널을 통해 개발될 수 있다. 이는 SCTP 데이터 채널과 RTC 4103 실시간 텍스트 같은 IETF 프로토콜 통해 향후 표준 게이트웨이가 활성화 될 때까지 충분할 것으로 보인다. 국제적으로 표준 RTT를 효과적이고 일관성 있게 지원하기 위해 W3C 그룹의 관련 작업과 협력 속에서 IETF에서 정의될 것이다.

A. 감사의 말

표준문서 편집자 들은 워킹 그룹 의장과 팀원들, 해럴드 앨베스트랜드, 스타판 호칸슨, 에릭 라거웨이와 도미니크 해자엘-매시우스에게 그들의 지원에 감사의 말을 전하고 싶다. 이 표준 규격의 실제적인 텍스트는 마틴 톰슨, 해럴드 앨베스트랜드, 저스틴 우버티, 에릭 리스콜라, 피터 대처, 얀이바르 브루아로이와 피터 세인트-안드레를 포함한 많은 사람들이 제공한 것이다. 댄 버넷은 이 표준 규격을 개발하는 동안 Voxeo와 Aspect로부터 받은 특별한 지원에 감사의 뜻을 전한다.

RTCRtpSender와 RTCRtpReceiver 객체는 W3C ORTC CG에 의해 처음 설명된 것이며, 이 규격에서 사용할 수 있도록 수정되었다.

B. 참조 문헌

(1) 규범적 참조문헌

[BUNDLE]

Negotiating Media Multiplexing Using the Session Description Protocol (SDP). C. Holmberg; H. Alvestrand; C. Jennings. IETF. 31 August 2017. Active Internet-Draft. URL: https://tools.ietf.org/html/draft-ietf-mmusic-sdp-bundle-negotiation

[DOM]

DOM Standard. Anne van Kesteren. WHATWG. Living Standard.
URL: https://dom.spec.whatwg.org/

[ECMASCRIPT-6.0]

ECMA-262 6th Edition, The ECMAScript 2015 Language Specification. Allen Wirfs-Brock. Ecma International. June 2015. Standard.
URL: http://www.ecma-international.org/ecma-262/6.0/index.html

[Fetch]

Fetch Standard. Anne van Kesteren. WHATWG. Living Standard.
URL: https://fetch.spec.whatwg.org/

[FILEAPI]

File API. Marijn Kruisselbrink; Arun Ranganathan. W3C. 11 September 2019. W3C Working Draft. URL: https://www.w3.org/TR/FileAPI/

[FIPS-180-4]

FIPS PUB 180-4 Secure Hash Standard. U.S. Department of Commerce/National Institute of Standards and Technology.
URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[GETUSERMEDIA]

Media Capture and Streams. Daniel Burnett; Adam Bergkvist; Cullen Jennings; Anant Narayanan; Bernard Aboba; Jan-Ivar Bruaroey; Henrik Boström. W3C. 2 July 2019. W3C Candidate Recommendation.
URL: https://www.w3.org/TR/mediacapture-streams/

[hr-time]

High Resolution Time Level 2. Ilya Grigorik. W3C. 21 November 2019. W3C Recommendation. URL: https://www.w3.org/TR/hr-time-2/

[HTML]

HTML Standard. Anne van Kesteren; Domenic Denicola; Ian Hickson; Philip Jägenstedt; Simon Pieters. WHATWG. Living Standard.
URL: https://html.spec.whatwg.org/multipage/

[IANA-HASH-FUNCTION]

Hash Function Textual Names. IANA.
URL: https://www.iana.org/assignments/hash-function-text-names/hash-function-text-names.xml

[IANA-RTP-2]

RTP Payload Format media types. IANA.
URL: https://www.iana.org/assignments/rtp-parameters/rtp-parameters.xhtml#rtp-parameters-2

[ICE]

Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. J. Rosenberg. IETF. April 2010. Proposed Standard. URL: https://tools.ietf.org/html/rfc5245

[INFRA]

Infra Standard. Anne van Kesteren; Domenic Denicola. WHATWG. Living Standard.
URL: https://infra.spec.whatwg.org/

[JSEP]

Javascript Session Establishment Protocol. Justin Uberti; Cullen Jennings; Eric Rescorla. IETF. 22 October 2018. Active Internet-Draft.
URL: https://tools.ietf.org/html/draft-ietf-rtcweb-jsep/

[MMUSIC-RID]

RTP Payload Format Restrictions. Adam Roach. IETF. 15 May 2018. Active Internet-Draft. URL: https://tools.ietf.org/html/draft-ietf-mmusic-rid/

[MMUSIC-SIMULCAST]

Using Simulcast in SDP and RTP Sessions. Bo Burman; Magnus Westerlund; Suhas Nandakumar; Mo Zanaty. IETF. 27 June 2018. Active Internet-Draft.
URL: https://tools.ietf.org/html/draft-ietf-mmusic-sdp-simulcast/

[RFC2119]

Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[RFC3550]

RTP: A Transport Protocol for Real-Time Applications. H. Schulzrinne; S. Casner; R. Frederick; V. Jacobson. IETF. July 2003. Internet Standard.
URL: https://tools.ietf.org/html/rfc3550

[RFC3890]

A Transport Independent Bandwidth Modifier for the Session Description Protocol (SDP). M. Westerlund. IETF. September 2004. Proposed Standard.
URL: https://tools.ietf.org/html/rfc3890

[RFC3986]

Uniform Resource Identifier (URI): Generic Syntax. T. Berners-Lee; R. Fielding; L. Masinter. IETF. January 2005. Internet Standard.
URL: https://tools.ietf.org/html/rfc3986

[RFC4566]

SDP: Session Description Protocol. M. Handley; V. Jacobson; C. Perkins. IETF. July 2006. Proposed Standard. URL: https://tools.ietf.org/html/rfc4566

[RFC4572]

Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP). J. Lennox. IETF. July 2006. Proposed Standard. URL: https://tools.ietf.org/html/rfc4572

[RFC5246]

The Transport Layer Security (TLS) Protocol Version 1.2. T. Dierks; E. Rescorla. IETF. August 2008. Proposed Standard. URL: https://tools.ietf.org/html/rfc5246

[RFC5285]

A General Mechanism for RTP Header Extensions. D. Singer; H. Desineni. IETF. July 2008. Proposed Standard. URL: https://tools.ietf.org/html/rfc5285

[RFC5389]

Session Traversal Utilities for NAT (STUN). J. Rosenberg; R. Mahy; P. Matthews; D. Wing. IETF. October 2008. Proposed Standard.
URL: https://tools.ietf.org/html/rfc5389

[RFC5506]

Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences. I. Johansson; M. Westerlund. IETF. April 2009. Proposed Standard. URL: https://tools.ietf.org/html/rfc5506

[RFC5888]

The Session Description Protocol (SDP) Grouping Framework. G. Camarillo; H. Schulzrinne. IETF. June 2010. Proposed Standard.
URL: https://tools.ietf.org/html/rfc5888

[RFC6464]

A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication. J. Lennox, Ed.; E. Ivov; E. Marocco. IETF. December 2011. Proposed Standard. URL: https://tools.ietf.org/html/rfc6464

[RFC6465]

A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication. E. Ivov, Ed.; E. Marocco, Ed.; J. Lennox. IETF. December 2011. Proposed Standard. URL: https://tools.ietf.org/html/rfc6465

[RFC6544]

TCP Candidates with Interactive Connectivity Establishment (ICE). J. Rosenberg; A. Keranen; B. B. Lowekamp; A. B. Roach. IETF. March 2012. Proposed Standard. URL: https://tools.ietf.org/html/rfc6544

[RFC7064]

URI Scheme for the Session Traversal Utilities for NAT (STUN) Protocol. S. Nandakumar;

G. Salgueiro; P. Jones; M. Petit-Huguenin. IETF. November 2013. Proposed Standard. URL: https://tools.ietf.org/html/rfc7064

[RFC7065]

Traversal Using Relays around NAT (TURN) Uniform Resource Identifiers. M. Petit-Huguenin; S. Nandakumar; G. Salgueiro; P. Jones. IETF. November 2013. Proposed Standard. URL: https://tools.ietf.org/html/rfc7065

[RFC7656]

A Taxonomy of Semantics and Mechanisms for Real-Time Transport Protocol (RTP) Sources. J. Lennox; K. Gross; S. Nandakumar; G. Salgueiro; B. Burman, Ed.. IETF. November 2015. Informational. URL: https://tools.ietf.org/html/rfc7656

[RFC7675]

Session Traversal Utilities for NAT (STUN) Usage for Consent Freshness. M. Perumal; D. Wing; R. Ravindranath; T. Reddy; M. Thomson. IETF. October 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7675

[RFC8174]

Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://tools.ietf.org/html/rfc8174

[RFC8261]

Datagram Transport Layer Security (DTLS) Encapsulation of SCTP Packets. M. Tuexen; R. Stewart; R. Jesup; S. Loreto. IETF. November 2017. Proposed Standard. URL: https://tools.ietf.org/html/rfc8261

[RFC8445]

Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal. A. Keranen; C. Holmberg; J. Rosenberg. IETF. July 2018. Proposed Standard. URL: https://tools.ietf.org/html/rfc8445

[RTCWEB-AUDIO]

WebRTC Audio Codec and Processing Requirements. JM. Valin; C. Bran. IETF. May 2016. Proposed Standard. URL: https://tools.ietf.org/html/rfc7874

[RTCWEB-DATA]

RTCWeb Data Channels. R. Jesup; S. Loreto; M. Tuexen. IETF. 14 October 2015. Active Internet-Draft.
URL: https://tools.ietf.org/html/draft-ietf-rtcweb-data-channel

[RTCWEB-DATA-PROTOCOL]

RTCWeb Data Channel Protocol. R. Jesup; S. Loreto; M. Tuexen. IETF. 14 October 2015. Active Internet-Draft.
URL: https://tools.ietf.org/html/draft-ietf-rtcweb-data- protocol

[RTCWEB-RTP]

Web Real-Time Communication (WebRTC): Media Transport and Use of RTP. C. Perkins; M. Westerlund; J. Ott. IETF. 17 March 2016. Active Internet-Draft.
URL: https://tools.ietf.org/html/draft-ietf-rtcweb-rtp-usage

[RTCWEB-SECURITY]

Security Considerations for WebRTC. Eric Rescorla. IETF. 22 January 2014. Active Internet-Draft. URL: https://tools.ietf.org/html/draft-ietf-rtcweb-security

[RTCWEB-TRANSPORT]

Transports for RTCWEB. H. Alvestrand. IETF. 31 October 2016. Active Internet-Draft.
URL: https://tools.ietf.org/html/draft-ietf-rtcweb-transports

[SCTP-SDP]

Session Description Protocol (SDP) Offer/Answer Procedures For Stream Control Transmission Protocol (SCTP) over Datagram Transport Layer Security (DTLS) Transport. C. Holmberg; R. Shpount; S. Loreto; G. Camarillo. IETF. 20 March 2017. Active Internet-Draft.
URL: https://tools.ietf.org/html/draft-ietf-mmusic-sctp-sdp

[SDP]

An Offer/Answer Model with Session Description Protocol (SDP). J. Rosenberg; H. Schulzrinne. IETF. June 2002. Proposed Standard.
URL: https://tools.ietf.org/html/rfc3264

[STUN-PARAMETERS]

STUN Error Codes. IETF. IANA. April 2011. IANA Parameter Assignment.
URL: https://www.iana.org/assignments/stun-parameters/stun-parameters.xhtml#stun-parameters-6

[TRICKLE-ICE]

Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol. E. Ivov; E. Rescorla; J. Uberti. IETF. 20 July 2015. Internet Draft (work in progress).
URL: http://datatracker.ietf.org/doc/draft-ietf-mmusic-trickle-ice

[WebCryptoAPI]

Web Cryptography API. Mark Watson. W3C. 26 January 2017. W3C Recommendation.
URL: https://www.w3.org/TR/WebCryptoAPI/

[WEBIDL]

Web IDL. Boris Zbarsky. W3C. 15 December 2016. W3C Editor's Draft.
URL: https://heycam.github.io/webidl/

[WEBRTC-STATS]

Identifiers for WebRTC's Statistics API. Harald Alvestrand; Varun Singh. W3C. 14 January 2020. W3C Candidate Recommendation.
URL: https://www.w3.org/TR/webrtc-stats/

[X509V3]

ITU-T Recommendation X.509 version 3 (1997). "Information Technology - Open Systems Interconnection - The Directory Authentication Framework" ISO/IEC 9594-8:1997.. ITU.

[X690]

Recommendation X.690 — Information Technology — ASN.1 Encoding Rules — Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER). ITU.
URL: https://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf

(2) 비규범적 참조문헌

[API-DESIGN-PRINCIPLES]

Web Platform Design Principles. Sangwhan Moon. 4 June 2020.
URL: https://w3ctag.github.io/design-principles/

[INDEXEDDB]

Indexed Database API. Nikunj Mehta; Jonas Sicking; Eliot Graff; Andrei Popescu; Jeremy Orlow; Joshua Bell. W3C. 8 January 2015. W3C Recommendation.
URL: https://www.w3.org/TR/IndexedDB/

[RFC4103]

RTP Payload for Text Conversation. G. Hellstrom; P. Jones. IETF. June 2005. Proposed Standard. URL: https://tools.ietf.org/html/rfc4103

[RFC6236]

Negotiation of Generic Image Attributes in the Session Description Protocol (SDP). I. Johansson; K. Jung. IETF. May 2011. Proposed Standard.
URL: https://tools.ietf.org/html/rfc6236

[RTCWEB-IP-HANDLING]

WebRTC IP Address Handling Recommendations. Guo-wei Shieh; Justin Uberti. IETF. 20 March 2016. Active Internet-Draft.
URL: https://tools.ietf.org/html/draft-ietf-rtcweb-ip-handling

[RTCWEB-OVERVIEW]

Overview: Real Time Protocols for Brower-based Applications. H. Alvestrand. IETF. 14 February 2014. Active Internet-Draft.
URL: https://tools.ietf.org/html/draft-ietf-rtcweb-overview

[RTCWEB-SECURITY-ARCH]

WebRTC Security Architecture. Eric Rescorla. IETF. 10 December 2016. Active Internet-Draft. URL: https://tools.ietf.org/html/draft-ietf-rtcweb-security-arch

[SDP-SIMULCAST]

Using Simulcast in SDP and RTP Sessions. B. Burman; M. Westerlund; S. Nandakumar; M. Zanaty. IETF. 27 June 2018. Active Internet-Draft.

URL: https://tools.ietf.org/html/draft-ietf-mmusic-sdp-simulcast

[xhr]

XMLHttpRequest Standard. Anne van Kesteren. WHATWG. Living Standard. URL: https://xhr.spec.whatwg.org/

1. 본 「W3C WebRTC 표준해설서」는 정부(과학기술정보통신부)의 재원으로, 정보통신기획평가원의 지원을 받은 과제(2017-0-00060, 사실표준화기구 전략대응 및 국제표준화전문가 활동강화)연구결과로 발간된 자료입니다.

◪ 총 괄 책 임 자 : 구경철(TTA 표준화본부장)
◪ 사 업 책 임 자 : 김대중(TTA 표준기획단장)
◪ 작성 및 검수자 : 손성영(유프리즘), 김찬성(아더윅스), 이문영(LG전자), 임주영(와이피랩스), 황대환(지나웍스)
◪ 표 준 기 획 단 : 김영재, 김정헌, 전철기, 진수경, 전지윤, 임영선, 오지훈

웹에서의 실시간 비디오, 오디오, 데이터통신 프로그래밍을 위한

W3C WebRTC 표준해설서 (Ver.2020)

2020년도 12월 인쇄
2020년도 12월 발행

발 행 소 : 한국정보통신기술협회
발 행 인 : 최 영 해
발간번호 : TTA-20113-SD
인 쇄 인 : (주)명진씨엔피 (02-2164-3000)

한국정보통신기술협회
Telecommunications Technology Association

463-824, 경기도 성남시 분당구 분당로 47
Tel : 031-780-9178, Fax : 031-724-0109
http://www.tta.or.kr