WebRTC ABR Player

Пример демонстрирует проигрывание потока, опубликованного на WCS сервере по WebRTC в нескольких качествах видео.

На скриншоте ниже:

- `Server url` - Websocket URL WCS сервера
- `Stream name` - имя потока
- `h`, `s`, `m send` - кнопки переключения принимаемого качества по именам профилей из файла `/usr/local/FlashphonerWebCallServer/conf/wcs_sfu_bridge_profiles.yml`

.

> **Attention**
>
> аудио дорожка проигрывается в отдельном элементе `audio` на странице

Исходный код примера

Исходный код разбит на следующие модули:

- `player.html` - HTML страница
- `player.css` - стили HTML страницы
- `player.js` - основная логика приложения

Анализ исходного кода

Для работы с исходным кодом примера возьмем версию файла `player.js`, доступную здесь

1. Локальные переменные

Объявление локальных переменных для работы с константами, SFU SDK, для отображения видео и работы с конфигурацией клиента

code

```
const constants = SFU.constants;
const sfu = SFU;
const PRELOADER_URL="../commons/media/silence.mp3"
```

2. Объект для хранения текущего состояния проигрывания

Хранит данные Websocket сессии, WebRTC соединения, SFU комнаты и объекта для отображения аудио и видео

code

```
const CurrentState = function() {
    let state = {
        pc: null,
        session: null,
        room: null,
        remoteDisplay: null,
        set: function(pc, session, room) {
            state.pc = pc;
            state.session = session;
            state.room = room;
        },
        clear: function() {
            state.room = null;
            state.session = null;
            state.pc = null;
        },
        setDisplay: function(display) {
            state.remoteDisplay = display;
        },
        disposeDisplay: function() {
            if (state.remoteDisplay) {
                state.remoteDisplay.stop();
                state.remoteDisplay = null;
            }
        }
    };
    return state;
}
```

3. Инициализация

`init()` code

Функция `init()` вызывается после того, как страница загрузится:

- инициализирует объекты состояния
- инициализирует поля ввода

```
const init = function() {
    $("#playBtn").prop('disabled', true);
```

```
    $("#url").prop('disabled', true);
    $("#streamName").prop('disabled', true);
    onDisconnected(CurrentState());
    $("#url").val(setURL());
}
```

4. Соединение с сервером

`connect()`, `SFU.createRoom()` code

Функция `connect()` вызывается по нажатию кнопки `Play` и делает следующее:

- создает объект `PeerConnection`
- очищает отображение статуса предыдущей сессии
- настраивает конфигурацию комнаты и создает Websocket сессию
- подписывается на события Websocket сессии

```
const connect = function(state) {
    // Create peer connection
    let pc = new RTCPeerConnection();
    // Create a config to connect to SFU room
    const roomConfig = {
        // Server websocket URL
        url: $("#url").val(),
        // Use stream name as room name to play ABR
        roomName: $("#streamName").val(),
        // Make a random participant name from stream name
        nickname: "Player-" + $("#streamName").val() + "-" + createUUID(4),
        // Set room pin
        pin: 123456
    }
    // Clean state display items
    setStatus("playStatus", "");
    setStatus("playErrorInfo", "");
    // Connect to the server (room should already exist)
    const session = sfu.createRoom(roomConfig);
    session.on(constants.SFU_EVENT.CONNECTED, function() {
        state.set(pc, session, session.room());
        onConnected(state);
        setStatus("playStatus", "CONNECTING...", "black");
    }).on(constants.SFU_EVENT.DISCONNECTED, function() {
        state.clear();
        onDisconnected(state);
        setStatus("playStatus", "DISCONNECTED", "green");
    }).on(constants.SFU_EVENT.FAILED, function(e) {
        state.clear();
        onDisconnected(state);
        setStatus("playStatus", "FAILED", "red");
        setStatus("playErrorInfo", e.status + " " + e.statusText, "red");
    });
}
```

5. Запуск проигрывания при установке соединения

`onConnected()` code

Функция `onConnected()`:

- настраивает действия по нажатию кнопки Stop
- подписывается на событие `SFU_ROOM_EVENT.PARTICIPANT_LIST` для проверки, опубликован ли поток в SFU комнате
- подписывается на события об ошибках комнаты
- вызывает функцию проигрывания

```
const onConnected = function(state) {
    $("#playBtn").text("Stop").off('click').click(function () {
        onStopClick(state);
    }).prop('disabled', false);
    $('#url').prop('disabled', true);
    $("#streamName").prop('disabled', true);
    // Add room event handling
    state.room.on(constants.SFU_ROOM_EVENT.PARTICIPANT_LIST, function(e) {
        // If the room is empty, the stream is not published yet
        if(!e.participants || e.participants.length === 0) {
            setStatus("playErrorInfo", "ABR stream is not published", "red");
            onStopClick(state);
        }
        else {
            setStatus("playStatus", "ESTABLISHED", "green");
            $("#placeholder").hide();
        }
    }).on(constants.SFU_ROOM_EVENT.FAILED, function(e) {
        // Display error state
        setStatus("playErrorInfo", e, "red");
    }).on(constants.SFU_ROOM_EVENT.OPERATION_FAILED, function (e) {
        // Display the operation failed
        setStatus("playErrorInfo", e.operation + " failed: " + e.error,
"red");
    }).on(constants.SFU_ROOM_EVENT.ENDED, function () {
        // Publishing is stopped, dispose playback and close connection
        setStatus("playErrorInfo", "ABR stream is stopped", "red");
        onStopClick(state);
    });
    playStreams(state);
}
```

6. Проигрывание потоков

`playStreams()`, `Room.join()` code

Функция `playStreams()`:

- инициализирует базовый элемент для отображения входящих медиа потоков

- настраивает WebRTC соединение в SFU комнате

```
const playStreams = function(state) {
    // Create remote display item to show remote streams
    state.setDisplay(initRemoteDisplay({
        div: document.getElementById("remoteVideo"),
        room: state.room,
        peerConnection: state.pc,
        displayOptions: {
            publisher: false,
            quality: true
        }
    }));
    state.room.join(state.pc);
}
```

7. Остановка проигрывания

`stopStreams()`, `CurrentState.disposeDisplay()` code

```
const stopStreams = function(state) {
    state.disposeDisplay();
}
```

8. Действия по нажатию кнопки `Play`

`onStartClick()`, `playFirstSound()`, `connect()` code

Функция `onStartClick()`:

- проверяет правильность заполнения полей ввода
- перед стартом воспроизведения, в браузере Safari вызывает функцию `playFirstSound()` для автоматического проигрывания аудио
- вызывает функцию `connect()`

```
const onStartClick = function(state) {
    if (validateForm("connectionForm")) {
        $("#playBtn").prop('disabled', true);
        if (Browser().isSafariWebRTC()) {
            playFirstSound(document.getElementById("main"),
PRELOADER_URL).then(function () {
                connect(state);
            });
        } else {
            connect(state);
        }
    }
}
```

9. Действия по нажатию кнопки `Stop`

`onStopClick()`, `Session.disconnect()` code

Функция `onStopClick()`:

- останавливает публикацию или воспроизведение
- разрывает Websocket сессию

```
const onStopClick = function(state) {
    $("#playBtn").prop('disabled', true);
    stopStreams(state);
    state.session.disconnect();
}
```

10. Действия при разрыве Websocket сессии

`onDisconnected()` code

Функция `onDisconnected()`:

- настраивает действия по нажатию кнопки `Play`
- открывает доступ к полям ввода `Server url` и `Room name`

```
const onDisconnected = function(state) {
    $("#placeholder").show();
    $("#playBtn").text("Play").off('click').click(function () {
        onStartClick(state);
    }).prop('disabled', false);
    $('#url').prop('disabled', false);
    $("#streamName").prop('disabled', false);
}
```