display.js - код для отображения публикуемого и получаемого видео и аудио

Данный модуль содержит код для отображения локального и получаемого видео и аудио. Код разделен на две области - для локального медиа и для получаемого медиа.

Отображение локального медиа

1. Функция-обертка

`initLocalDisplay()` code

Обертка для кода отображения локального медиа

```
const initLocalDisplay = function(localDisplayElement){
    ...
}
```

2. Локальные переменные

code

Объявление локальных переменных

```
const localDisplayDiv = localDisplayElement;
const localDisplays = {};
```

3. Удаление элемента локального отображения

removeLocalDisplay() code

Удаление локального элемента отображения при завершении публикации дорожки

```
const removeLocalDisplay = function(id) {
    delete localDisplays[id];
    $('#' + id).remove();
    reassembleLocalLayout();
}
```

4. Поиск видео элемента без аудио

`getAudioContainer()` code

Поиск видео элемента без аудио дорожки

```
const getAudioContainer = function() {
    for (const [key, value] of Object.entries(localDisplays)) {
        let video = value.getElementsByTagName("video");
        if (video && video[0]) {
            let audioStateButton = value.getElementsByTagName("button");
            let audioTracks = video[0].srcObject.getAudioTracks();
            if (!audioTracks || audioTracks.length === 0) {
                return {
                    id: value.id,
                    video: video[0],
                    audioStateDisplay: audioStateButton[0]
                }
            }
        }
    }
};
```

5. Добавление дорожки для локального отображения

`add()` code

Функция добавляет дорожку для локального отображения

add

Проверка, какая дорожка добавляется. Если аудио, поиск ближайшего видео элемента без аудио, и добавление аудио дорожки к нему

code

```
if (stream.getAudioTracks().length > 0) {
    let videoElement = getAudioContainer();
    if (videoElement) {
        let track = stream.getAudioTracks()[0];
        videoElement.video.srcObject.addTrack(track);
        videoElement.audioStateDisplay.innerHTML = "Audio state: " +
stream.getAudioTracks()[0].enabled;
        track.addEventListener("ended", function() {
            videoElement.video.srcObject.removeTrack(track);
            videoElement.audioStateDisplay.innerHTML = "Audio state: " +
false;
            //check video element has no tracks left
            for (const [key, vTrack] of
Object.entries(videoElement.video.srcObject.getTracks())) {
                if (vTrack.readyState !== "ended") {
                    return;
                }
            }
```

```
            removeLocalDisplay(videoElement.id);
        });
        return;
    }
}
```

Создание нового контейнера для отображения

code

```
const coreDisplay = document.createElement('div');
coreDisplay.setAttribute("style","width:200px; height:auto; border: solid;
border-width: 1px");
coreDisplay.id = stream.id;
```

Создание контейнера для отображения имени

code

```
const streamNameDisplay = document.createElement("div");
streamNameDisplay.innerHTML = "Name: " + name;
streamNameDisplay.setAttribute("style","width:auto; height:30px");
coreDisplay.appendChild(streamNameDisplay);
```

Создание элемента для отображения состояния аудио. Подписка на событие `click` для отключения/включения аудио.

code

```
const audioStateDisplay = document.createElement("button");
audioStateDisplay.setAttribute("style","width:auto; height:30px");
audioStateDisplay.innerHTML = "Audio state: " +
(stream.getAudioTracks().length > 0 ? stream.getAudioTracks()[0].enabled :
false);
audioStateDisplay.addEventListener('click', function(){
    if (stream.getAudioTracks().length > 0) {
        stream.getAudioTracks()[0].enabled = !(stream.getAudioTracks()
[0].enabled);
        audioStateDisplay.innerHTML = "Audio state: " +
stream.getAudioTracks()[0].enabled;
    }
});
coreDisplay.appendChild(audioStateDisplay);
```

Создание контейнера для отображения видео потока

code

```
const streamDisplay = document.createElement('div');
streamDisplay.id = id;
streamDisplay.setAttribute("style","width:auto; height:auto");
coreDisplay.appendChild(streamDisplay);
```

Создание видео элемента и добавление его в контейнер

code

```
const video = document.createElement("video");
streamDisplay.appendChild(video);
video.srcObject = stream;
video.muted = true;
video.onloadedmetadata = function (e) {
    video.play();
};
```

Подписка на событие `ended` для дорожки. Если дорожка завершилась, и не осталось ни одной активной дорожки, удаление контейнера

code

```
stream.getTracks().forEach(function(track){
    track.addEventListener("ended", function() {
        video.srcObject.removeTrack(track);
        //check video element has no tracks left
        for (const [key, vTrack] of
Object.entries(video.srcObject.getTracks())) {
            if (vTrack.readyState !== "ended") {
                return;
            }
        }
        removeLocalDisplay(id);
    });
});
```

Подписка на событие `resize`, чтобы масштабировать видео под размеры контейнера.

code

```
video.addEventListener('resize', function (event) {
    streamNameDisplay.innerHTML = "Name: " + name + " " + video.videoWidth +
"x" + video.videoHeight;
    resizeVideo(event.target);
});
```

Сохранение контейнера, обновление отображения и возврат нового контейнера.

code

```
localDisplays[id] = coreDisplay;
reassembleLocalLayout();
return coreDisplay;
```

6. Обновление локального отображения на странице

`reassembleLocalLayout()` code

Вспомогательная функция пересчитывает сетку локальных контейнеров и перерисовывает их на странице

```
const reassembleLocalLayout = function() {
    let gridWidth = gridSize(Object.keys(localDisplays).length).x;
    let container = document.createElement('div');
    let row;
    let rowI = 1;
    let colI = 0;
    for (const [key, value] of Object.entries(localDisplays)) {
        if (row) {
            if (colI >= gridWidth) {
                row = createRow(container);
                rowI++;
                colI = 0;
            }
        } else {
            row = createRow(container);
        }
        $("#" + key).detach();
        let col = createCol(row);
        col.appendChild(value);
        colI++;
    }
    $(localDisplayDiv).empty();
    localDisplayDiv.appendChild(container);
}
```

7. Экспорт функции для использования в основном коде

code

```
return {
    add: add
}
```

Отображение получаемого медиа

1. Функция-обертка

`initRemoteDisplay()` code

Обертка для кода отображения получаемого медиа

```
const initRemoteDisplay = function(room, mainDiv, peerConnection) {
    ...
}
```

2. Локальные переменные

code

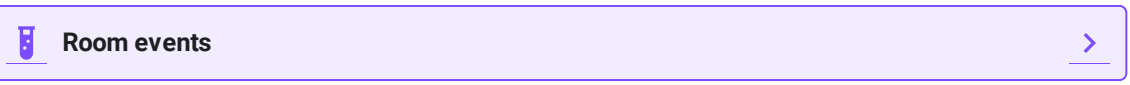
Объявление локальных переменных

```
const constants = SFU.constants;
const remoteParticipants = {};
```

3. Подписка на события комнаты

code

Подписка на необходимые события комнаты



SFU_ROOM_EVENT.ADD_TRACKS

Поиск участника. Если не найден, создание нового участника

code

```
let participant = remoteParticipants[e.info.nickName];
if (!participant) {
    participant = {};
    participant.nickName = e.info.nickName;
    participant.tracks = [];
    participant.displays = [];
    remoteParticipants[participant.nickName] = participant;
}
```

Добавление новых дорожек для этого участника

code

```
participant.tracks.push.apply(participant.tracks, e.info.info);
```

Создание контейнера для каждой дорожки

code

```
for (const pTrack of e.info.info) {
    let createDisplay = true;
    for (let i = 0; i < participant.displays.length; i++) {
        let display = participant.displays[i];
        if (pTrack.type === "VIDEO") {
            if (display.hasVideo()) {
                continue;
```

```
            }
            display.videoMid = pTrack.mid;
            display.setTrackInfo(pTrack);
            createDisplay = false;
            break;
        } else if (pTrack.type === "AUDIO") {
            if (display.hasAudio()) {
                continue;
            }
            display.audioMid = pTrack.mid;
            createDisplay = false;
            break;
        }
    }
    if (!createDisplay) {
        continue;
    }
    let display = createRemoteDisplay(participant.nickName,
participant.nickName, mainDiv);
    participant.displays.push(display);
    if (pTrack.type === "VIDEO") {
        display.videoMid = pTrack.mid;
        display.setTrackInfo(pTrack);
    } else if (pTrack.type === "AUDIO") {
        display.audioMid = pTrack.mid;
    }
}
```

SFU_ROOM_EVENT.REMOVE_TRACKS

Поиск участника. Если не найден, возврат

code

```
const participant = remoteParticipants[e.info.nickName];
if (!participant) {
    return;
}
```

Перебор дорожек участника

code

```
for (const rTrack of e.info.info) {
```

Поиск и удаление дорожки с таким же mid, какой был получен в событии

code

```
for (let i = 0; i < participant.tracks.length; i++) {
    if (rTrack.mid === participant.tracks[i].mid) {
        participant.tracks.splice(i, 1);
        break;
```

```
    }
}
```

Поиск контейнера, в котором проигрывается дорожка, и удаление дорожки. Если в контейнере не осталось дорожек, он также удаляется

code

```
for (let i = 0; i < participant.displays.length; i++) {
    let found = false;
    const display = participant.displays[i];
    if (display.mids.audio.includes(rTrack.mid)) {
        //remove from mids array
        display.mids.audio.splice(display.mids.audio.indexOf(rTrack.mid), 1);
        //stop track and remove stream
        display.audioStreams[rTrack.mid].getAudioTracks()[0].stop();
        delete display.audioStreams[rTrack.mid];
        //remove audio element
        display.display.removeChild(display.audioElements[rTrack.mid]);
        delete display.audioElements[rTrack.mid];
        found = true;
    } else if (display.mids.video === rTrack.mid) {
        display.mids.video = undefined;
        display.mediaStream.getVideoTracks()[0].stop();
        found = true;
    }
    if (display.mids.audio.length === 0 && display.mids.video === undefined)
{
        const video = display.display.getElementsByTagName("video")[0]
        video.pause();
        video.srcObject = null;
        display.display.remove();
        participant.displays.splice(i, 1);
    }
    if (found) {
        break;
    }
}
```

SFU_ROOM_EVENT.LEFT

Поиск и удаление участника

code

```
let participant = remoteParticipants[e.name];
if (!participant) {
    return;
}
participant.displays.forEach(function(display){
    display.dispose();
})
delete remoteParticipants[e.name];
```

SFU_ROOM_EVENT.TRACK_QUALITY_STATE

Поиск участника. Если не найден, возврат

code

```
console.log("Received track quality state");
const participant = remoteParticipants[e.info.nickName];
if (!participant) {
    return;
}
```

Перебор дорожек участника

code

```
for (const rTrack of e.info.tracks) {
    ...
}
```

Поиск соответствующего контейнера и обновление качества

code

```
const mid = rTrack.mid;
for (let i = 0; i < participant.displays.length; i++) {
    const display = participant.displays[i];
    if (display.videoMid === mid) {
        display.updateQualityInfo(rTrack.quality);
        break;
    }
}
```

4. Создание контейнера для отображения принимаемых потоков

`createRemoteDisplay()` code

Вспомогательная функция создает контейнер на основе данных о потоке и дорожках

createRemoteDisplay

Удаление контейнера

`dispose()` code

```
dispose: function() {
    cell.remove();
```

```
}
```

Скрытие контейнера

`hide()` code

```
hide: function(value) {
    if (value) {
        cell.style.display = "none";
    } else {
        cell.style.display = "block";
    }
}
```

Создание аудио элемента

`setAudio()` code

```
setAudio: function(stream) {
    if (audio) {
        audio.remove();
    }
    if (!stream) {
        audio = null;
        this.audioMid = undefined;
        return;
    }
    audio = document.createElement("audio");
    audio.controls = "controls";
    cell.appendChild(audio);
    audio.srcObject = stream;
    audio.play();
}
```

Создание видео элемента

`setVideo()` code

```
setVideo: function(stream) {
    if (video) {
        video.remove();
    }

    if (stream == null) {
        video = null;
        this.videoMid = undefined;
        qualityDivs.forEach(function(div) {
            div.remove();
        });
        qualityDivs = [];
        tidDivs.forEach(function(div) {
            div.remove();
        });
        tidDivs = [];
```

```
        return;
    }
    video = document.createElement("video");
    streamDisplay.appendChild(video);
    video.srcObject = stream;
    video.onloadedmetadata = function (e) {
        video.play();
    };
    video.addEventListener("resize", function (event) {
        streamNameDisplay.innerHTML = "Name: " + name + " " +
video.videoWidth + "x" + video.videoHeight;
        resizeVideo(event.target);
    });
}
```

Отображение информации о качестве

`setTrackInfo()` code

```
setTrackInfo: function(trackInfo) {
    if (trackInfo && trackInfo.quality) {
        for (let i = 0; i < trackInfo.quality.length; i++) {
            const qualityDiv = document.createElement("button");
            qualityDivs.push(qualityDiv);
            qualityDiv.innerText = trackInfo.quality[i];
            qualityDiv.setAttribute("style", "display:inline-block; border:
solid; border-width: 1px");
            qualityDiv.style.color = "red";
            qualityDiv.addEventListener('click', function(){
                console.log("Clicked on quality " + trackInfo.quality[i] + "
trackId " + trackInfo.id);
                if (qualityDiv.style.color === "red") {
                    return;
                }
                for (let c = 0; c < qualityDivs.length; c++) {
                    if (qualityDivs[c].style.color !== "red") {
                        qualityDivs[c].style.color = "gray";
                    }
                }
                qualityDiv.style.color = "blue";
                room.changeQuality(trackInfo.id, trackInfo.quality[i]);
            });
            qualityDisplay.appendChild(qualityDiv);
        }
        for (let i = 0; i < 3; i++) {
            const tidDiv = document.createElement("button");
            tidDivs.push(tidDiv);
            tidDiv.innerText = "TID"+i;
            tidDiv.setAttribute("style", "display:inline-block; border:
solid; border-width: 1px");
            tidDiv.style.color = "gray";
            tidDiv.addEventListener('click', function(){
                console.log("Clicked on TID " + i + " trackId " +
trackInfo.id);
                for (let c = 0; c < tidDivs.length; c++) {
                    tidDivs[c].style.color = "gray";
                }
```

```
                tidDiv.style.color = "blue";
                room.changeQuality(trackInfo.id, null, i);
            });
            tidDisplay.appendChild(tidDiv);
        }
    }
}
```

Обновление качества

`updateQualityInfo()` code

```
updateQualityInfo: function(videoQuality) {
    for (const qualityInfo of videoQuality) {
        for (const qualityDiv of qualityDivs) {
            if (qualityDiv.innerText === qualityInfo.quality){
                if (qualityInfo.available === true) {
                    qualityDiv.style.color = "gray";
                } else {
                    qualityDiv.style.color = "red";
                }
                break;
            }
        }
    }
}
```

5. Работа с `PeerConnection`

code

Подписка на событие `PeerConnection.ontrack`.

```
peerConnection.ontrack = ({transceiver}) => {
    let rParticipant;
    console.log("Attach remote track " + transceiver.receiver.track.id + "
kind " + transceiver.receiver.track.kind + " mid " + transceiver.mid);
    for (const [nickName, participant] of Object.entries(remoteParticipants))
{
        for (const pTrack of participant.tracks) {
            console.log("Participant " + participant.nickName + " track " +
pTrack.id + " mid " + pTrack.mid);
            if (pTrack.mid === transceiver.mid) {
                rParticipant = participant;
                break;
            }
        }
        if (rParticipant) {
            break;
        }
    }
    if (rParticipant) {
        for (const display of rParticipant.displays) {
            if (transceiver.receiver.track.kind === "video") {
```

```
                if (display.videoMid === transceiver.mid) {
                    let stream = new MediaStream();
                    stream.addTrack(transceiver.receiver.track);
                    display.setVideo(stream);
                    break;
                }
            } else if (transceiver.receiver.track.kind === "audio") {
                if (display.audioMid === transceiver.mid) {
                    let stream = new MediaStream();
                    stream.addTrack(transceiver.receiver.track);
                    display.setAudio(stream);
                    break;
                }
            }
        }
    } else {
        console.warn("Failed to find participant for track " +
transceiver.receiver.track.id);
    }
}
```

Поиск участника на основе mid дорожки

code

```
let rParticipant;
console.log("Attach remote track " + transceiver.receiver.track.id + " kind "
+ transceiver.receiver.track.kind + " mid " + transceiver.mid);
for (const [nickName, participant] of Object.entries(remoteParticipants)) {
    for (const pTrack of participant.tracks) {
        console.log("Participant " + participant.nickName + " track " +
pTrack.id + " mid " + pTrack.mid);
        if (pTrack.mid === transceiver.mid) {
            rParticipant = participant;
            break;
        }
    }
    if (rParticipant) {
        break;
    }
}
```

Поиск соответствующего контейнера и добавление к нему дорожки

code

```
for (const display of rParticipant.displays) {
    if (transceiver.receiver.track.kind === "video") {
        if (display.videoMid === transceiver.mid) {
            let stream = new MediaStream();
            stream.addTrack(transceiver.receiver.track);
            display.setVideo(stream);
            break;
        }
    } else if (transceiver.receiver.track.kind === "audio") {
```

```
        if (display.audioMid === transceiver.mid) {
            let stream = new MediaStream();
            stream.addTrack(transceiver.receiver.track);
            display.setAudio(stream);
            break;
        }
    }
}
```