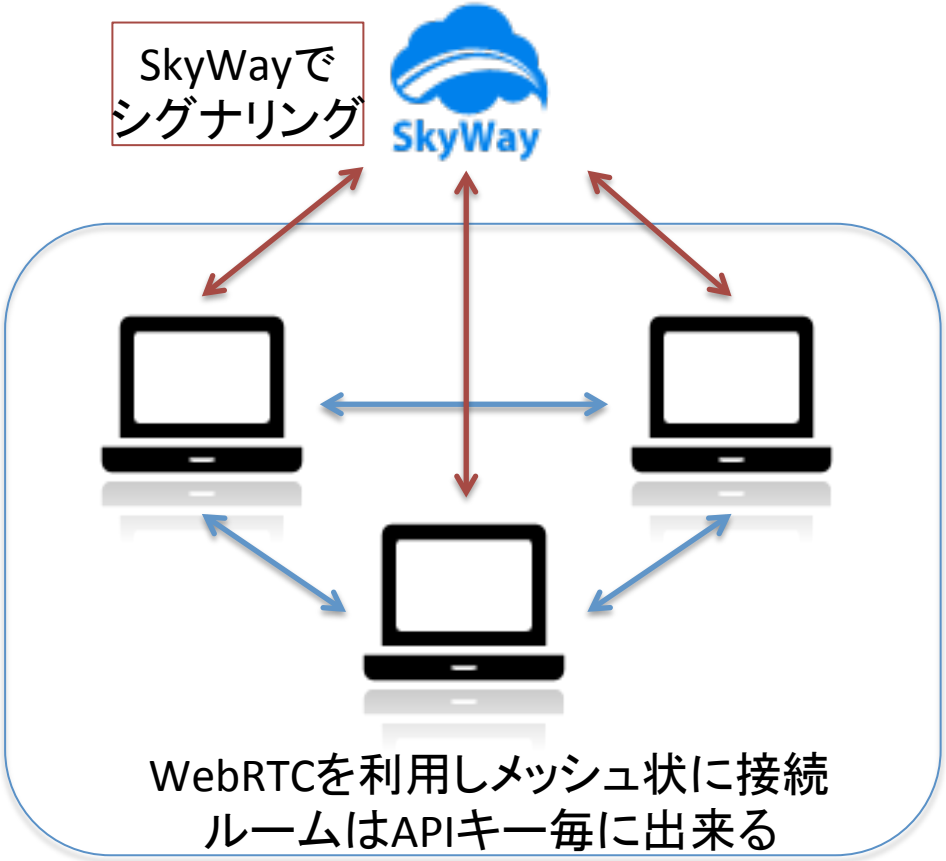
SkyWay ハンズオン

今日作るもの

簡単な多対多のビデオチャットアプリ



https://github.com/skyway/webrtc-handson-js

使ってみよう！

//skyway.io

Step 1-1

- script.js を好きなエディターで開く
- APIKEYを各自https://skyway.io/ds から取得したものを設定
- 利用可能ドメインはlocalhostに設定

```
// 定数宣言
var APIKEY = 'ad8276ac-56c9-436e-83ca-cbe9b38ec386';
```

Step 1-2

- getUserMediaは、WebRTCの仕様の一部で、カメラとマイクのデータを取得するためのメソッド
- WebRTCの機能はまだ新しいからブラウザ毎に違う名前が付けられている
  - いちいち対応が面倒くさいから同じ名前の関数にまとめる

```
// getUserMediaのcompatibility
navigator.getUserMedia =  navigator.getUserMedia ||
                          navigator.webkitGetUserMedia ||
                          navigator.mozGetUserMedia;
```

Step 2-1

- Peerクラスは、SkyWayが提供するシグナリングのためのクラス。
- コンストラクタを呼んで、Peerオブジェクトを生成し、シグナリングサーバに接続する

```
// Peerオブジェクトを生成
peer = new Peer({key: APIKEY, debug: 3});
```

Step 2-2

- エラーが起きたらコンソールに書き出す
  - APIキーが間違ってる
  - ドメインが登録されていないなど

```
// エラーハンドラ
peer.on('error', function(err){
    console.error(err);
});
```

Step 2-3

- シグナリングサーバに接続出来たら何かしらの処理を実行する
  - とりあえずコンソールにidを書き出してみよう

```
// openイベントのハンドラ
peer.on('open', function(id) {
    myPeerid = id;
    console.log('My peer ID is: ' + id);
});
```

Step 3

- 自分のカメラの映像を取得して表示する
  - getUserMediaのコールバックで、カメラの映像が取得できる。
  - URL.createObjectURLでURLの形式に変換し、video要素のsrc属性にセットする。

```
// openイベントのハンドラ
peer.on('open', function(id) {
    myPeerid = id;
    console.log('My peer ID is: ' + id);

    navigator.getUserMedia({
        audio: true,
        video: true
    }, function(stream){
        $('#myStream').prop('src', URL.createObjectURL(stream));
        myStream = stream;
    }, function(){
        console.error('getUserMedia error');
    });
});
```

Step 4

- 共通処理

```
// コールの追加
function addCall(call){
    callList.push(call);
}

// コールの削除
function removeCall(call){
    var position = callList.indexOf(call);
    if(position > 0){
        callList.splice(position,1)
    }
}

// VIDEO要素を追加する
function addVideo(call,stream){
    var videoDom = $('<video autoplay>');
    videoDom.attr('id',call.peer);
    videoDom.prop('src',URL.createObjectURL(stream));

    $('.videosContainer').append(videoDom);
}

// VIDEO要素を削除する
function removeVideo(call){
    $('#'+call.peer).remove();
}
```

Step 5-1

- コールがかかってきた時の処理

```
// callイベント用のハンドラを設置
peer.on('call', function(call) {
    // 相手からcallイベントがきたらstreamを送り返す（応答する）
    call.answer(myStream);

    // callオブジェクトのイベントをセット
    setupCallEventHandlers(call);

    // 接続先毎のcallオブジェクトをマルチパーティ管理用に保存
    addCall(call);

});
```

Step 5-2

- コールがかかってきた時の処理

```
// callオブジェクトのイベントをセットする
function setupCallEventHandlers(call){

    // 相手からstreamイベンがきたらそのstreamをVIDEO要素に表示する
    call.on('stream', function(stream){
        addVideo(call,stream);
    });

    // 相手からcloseイベントがきたらコネクションを切断して保存した
    // callオブジェクトを削除、対応するVIDEOS要素も削除
    call.on('close', function(){
        call.close();
        removeCall(call);
        removeVideo(call);
    });

}
```

Step 6

- 実際かける

```
// ユーザリストを取得して片っ端から繋ぐ
function connectToPeers() {
    peer.listAllPeers(function(list){
        for(var cnt = 0;cnt < list.length;cnt++){
            if(myPeerid != list[cnt]){
                var call = peer.call(list[cnt],myStream);
                setupCallEventHandlers(call);
                addCall(call);
            }
        }
    });
}
```

Step 6

- 実際かける

```
// openイベントのハンドラ
peer.on('open', function(id) {
    myPeerid = id;
    console.log('My peer ID is: ' + id);

    navigator.getUserMedia({
        audio: true,
        video: true
    }, function(stream){
        $('#myStream').prop('src', URL.createObjectURL(stream));
        myStream = stream;

        // 全ユーザと接続を行う
        connectToPeers();
    }, function(){
        console.error('getUserMedia error');
    });
});
```

チャレンジ課題

1. データチャンネルを利用してテキストチャットを実装する
   参考：https://goo.gl/tsRDNC

2. 受信したビデオのスクリーンショットを取る機能を実装する
   ヒント：canvas要素に書き出してから画像に変換する

3. 自分の映像を送信しない（映像オフ）機能を実装する
   ヒント：myStream のvideoTrackをいじる必要がある
   参考：https://goo.gl/dmzkuZ