第 6 回

チャット（双方向通信）とその応用

SNSにチャット機能（双方向通信）は欠かせません。
双方向通信を実現するには、ブラウザ側のアプリに加え、
サーバ側のアプリが必要です。
文字・画像・各種データをリアルタイムでやりとりする
チャットプログラミングを学びます。

フィジカルコンピューティング演習
愛知淑徳大学 人間情報学部　　國分 三輝

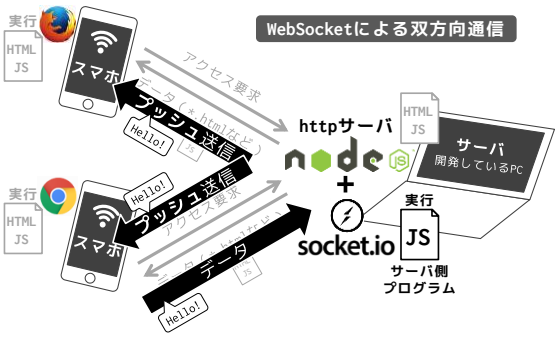
✈ 6.0 チャット（双方向通信）とその応用

SNS にチャット機能（双方向通信）は欠かせません。双方向通信はチャットだけでなく、オンラインゲームなどでも必須の機能です。双方向通信を実現するには、ここまで学んだ「クライアント（ブラウザ）が実行するアプリ（HTML・JavaScript）」に加え、「サーバ側で実行するアプリ」が必要です。文字・画像・各種データをリアルタイムでやりとりするチャットプログラミングを学びます。

6.1 シンプルなチャット

➔ 3 ページ

サイトに接続しているユーザにリアルタイムでテキストメッセージ（文字）を配信するシンプルなチャットアプリを作りながら、サーバとクライアントが連携したアプリの作り方を学びます。



6.2 おしゃべりなチャット（Chrome 限）

➔ 9 ページ

6.1 のチャットアプリを少しだけ改造して、音声合成でメッセージを読み上げる機能を付けます。

6.3 ぶるぶるチャット

➔ 11 ページ

テキストメッセージ以外のデータをリアルタイムに送るチャットアプリを作ります。

6.4 画像付きチャット

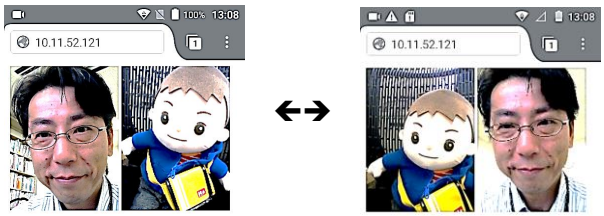
➔ 13 ページ

テキストメッセージと画像をあわせて送るチャットアプリを作りながら、複数のメディアをやりとりする方法を学びます。

6.5 Face to Face (Firefox 限)

➔ 17 ページ

カメラの映像 (動画) を送りあうビデオチャットを作りながら、ビデオ通話や遠隔監視の方法を学びます。



本日の課題

➔ 20 ページ

✈ 6.1 シンプルなチャット

	PC	And	iOS
Firefox	○	○	○
Chrome	○	○	○

サイトに接続しているユーザにリアルタイムでテキストメッセージ (文字) を配信するシンプルなチャットアプリを作りながら、サーバとクライアントが連携したアプリの作り方を学びます。

Input サーバ側プログラム (JavaScript) を入力

- **C:￥Users￥ (ユーザ名) ￥**の中に **6.1** というフォルダを作ります。
- 以下のコードを Brackets で入力し、6.1 フォルダ内に **app.js** という名前で保存します。

※ ①や②などの丸数字は打ち込まないこと
　コメント文は打ち込み必須ではありません

C:￥Users￥ (ユーザ名) ￥**6.1**￥ **app.js**

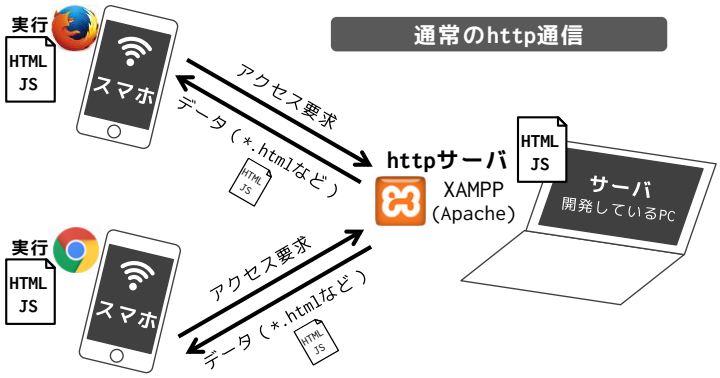
```
// チャットアプリのためのサーバ側プログラム
// サーバ PC に設定した Node.js というシステムがこの JavaScript を実行する                ①

'use strict';                                  // 厳格モードにする                        ②

// Node.js の http サーバ機能を使う ( 定型・おまじないと思っておいて構わない )            ③
const express = require('express');            // express モジュールを使う
const app = express();                         // express でアプリを作る
app.use(express.static(__dirname));            // ホーム dir にあるファイルを使えるようにする
app.get('/', function (req, res) {             // アクセス要求があったら
    res.sendFile(__dirname + '/index.html');    // index.html を送る
});
const server = require('http').Server(app); // http サーバを起動してアプリを実行
server.listen(80);                             // サーバの 80 番ポートでアクセスを待つ

const io = require('socket.io')(server);    // socket.io モジュールをサーバにつなぐ       ④

// クライアントから双方向通信 ( socket ) が来た時の処理
io.on('connection', function(socket) {      // socket 接続があって                       ⑤
    socket.on('all', function(data) {       // 'all'というイベントの socket が来たら      ⑥
        io.sockets.emit('msg', data);       // 'msg'というイベントで全員に data 配信      ⑦
    });
    socket.on('others', function(data) {    // 'others'というイベントの socket が来たら   ⑧
        socket.broadcast.emit('msg', data); // 'msg'というイベントで自分以外に data 配信  ⑨
    });
});
```

★この app.js は 6.2～6.5 でも同じものを使います。注意深く入力しましょう！

Learn 解　説

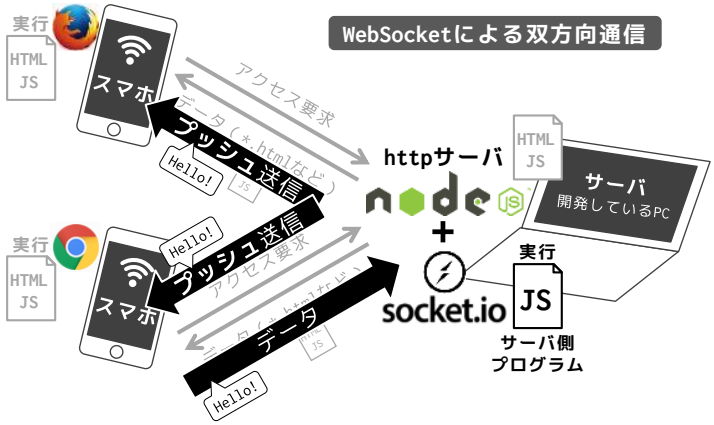
①　// サーバ PC に設定した Node.js というシステムがこの JavaScript を実行する

これまで用いてきた通常の http 通信のイメージを下図に示します。



サーバ PC で http サーバである XAMPP(Apache)を実行しておき、特定の場所に index.html や各種データを置きました。http サーバは、スマホなどのクライアントからアクセス要求 (データをくださいという要求) があったら、html ファイルなどのデータを要求元のブラウザに送ります。ブラウザは送られてきた html ファイル内のプログラムを実行します。

これに対して、これから作ろうとしているシステムのイメージを下図に示します。



http サーバとして XAMPP の代わりに Node.js という環境※1 を使います。Node.js は通常の http 通信 (ブラウザがプログラムを実行) も行いつつ、サーバ PC 側で別のプログラムを実行することができる環境です。この Node.js に socket.io というライブラリ※1 をつなげたプログラムを書くと、クライアントからアクセス要求が無くても、接続中のクライアントにデータをサーバ側からプッシュ送信することができます。LINE などでメッセージがすぐに届いたり通知が出るように、リアルタイムでデータを送受信することができるようになります。

上で作成した app.js は、この、サーバ側で実行されるプログラムです。

※1　サーバ PC への Node.js と socket.io の設定方法は付録 7・8 で紹介します。

② `'use strict';`

サーバ側プログラムは厳格さが求められます。`'use strict';` と書くと、JavaScript は厳密なエラーチェックの元で動作します。「サーバ側プログラムでは書いておく」と覚えておきましょう。

③ `// Node.js の http サーバ機能を使う`

Node.js の http サーバ機能を使うためのプログラムです。これらはほぼ決まった手順なので、あまり深く考えずに「おまじない」と思っておいて構いません。

④ `const io = require('socket.io')(server);`

③で起動させた Node.js の http サーバに、socket.io という双方向通信ライブラリをつないで、クライアントからデータが送られてくるのを待ちます。この部分も「おまじない」と思っておいて構いません。

⑤ `io.on('connection', function(socket) { ⑥⑦⑧⑨ } );`

複数行（8 行）にまたがっていますが、下記の⑤～⑧も含めて 1 行です。

`io.on('connection', function(socket) {...})`は「クライアントからの socket 接続があったら function(socket) {...} 内の処理を実行せよ」という意味です。

この場合、クライアントからの socket 接続があったら、以下の⑤（およびその中の⑥）または⑦（およびその中の⑦）を実行します。

⑥ `socket.on('all', function(data) { io.sockets.emit('msg', data); } );`

複数行にまたがっていますがこれで 1 行です。

`socket.on` は「送られてきた socket が'～'というイベント名だったら」という意味です。

この場合、socket が'all'というイベント名だったら、function(data) {...} 内の処理を実行します。なお、変数 data には送られてきたデータが自動で入ります。

⑦ `io.sockets.emit('msg', data);`

`io.sockets.emit();`は、接続している全てのクライアント（データ送付元も含む）にデータを送る命令です。この場合、'msg'というイベント名を付けて、送られてきた data をそのまま全クライアントに送ります。

⑧ `socket.on('others', function(data) { socket.broadcast.emit('msg', data); });`

⑤と同様、複数行にまたがっていますがこれで 1 行です。

この場合、'others'というイベント名だったら、function(data) {...} 内の処理を実行します。

⑨ `socket.broadcast.emit('msg', data);`

`socket.broadcast.emit();`は、送信元以外の全てのクライアントにデータを送る命令です。

この場合、'msg'というイベント名を付けて、送られてきた data をそのまま送信元以外のクライアントに送ります。

Input クライアント側プログラム (HTML・JavaScript) を入力

- **C:￥Users￥ (ユーザ名) ￥6.1** の中に 1.1 で作った index.html (テンプレート) をコピーし、Brackets で開き、以下のコードを入力しましょう。

※ ①や②などの丸数字は打ち込まないこと
　コメント文は打ち込み必須ではありません

C:￥Users￥ (ユーザ名) ￥6.1￥ **index.html**

```
( 略 )
        <title>6.1 シンプルなチャット</title>
    </head>
    <body>
        <input type='text' id='txt'>                    <!-- テキストボックス -->
        <input type='button' id='btn' value='送信'>     <!-- 送信ボタン -->
        <div id='msgArea'></div>                        <!-- メッセージ表示エリア -->
        <script src='/socket.io/socket.io.js'></script> <!-- socket.io の読み込み -->    ⑩
        <script>
            const socket = io();                // 双方向通信用のサーバに接続              ⑪

            // サーバへのデータ送信
            const btn = document.getElementById('btn');     // ボタンを取得
            btn.addEventListener('click', sendData);        // click で sendData を実行
            function sendData() {               // データの送信処理
                const txt = document.getElementById('txt'); // テキストボックスを取得
                if (txt.value == '') {          // テキストボックスが空なら
                    return;                     // 何もしないで抜ける
                }
                const data = {                  // 送信するデータオブジェクトを作る       ⑫
                    text: txt.value             // text というプロパティにテキストを入れる
                };
                socket.emit('all', data);       // 'all'というイベントで data を送信      ⑬
                txt.value = '';                 // テキストボックスを空にする
            }

            // サーバからのデータ受信
            socket.on('msg', function(data) {  // 'msg'というイベントの socket が来たら  ⑭
                receiveData(data);              // receiveData を実行
            });
            function receiveData(data) {        // データの受信処理
                const newMsg = document.createElement('div'); // 新規メッセージ要素作成 ⑮
                newMsg.innerHTML = data.text;  // newMsg に data の text プロパティを入れる
                const msgArea = document.getElementById('msgArea');   // 表示エリアを取得
                msgArea.appendChild(newMsg);    // 表示エリアに子要素として newMsg を追加 ⑯
                const sound = new Audio();      // Audio オブジェクトを作る               ⑰
                sound.src = 'mokkin.mp3';       // src 属性に効果音ファイルを指定
                sound.play();                   // 効果音を再生
            }
        </script>
( 以下略 )
```

Check PC とスマホで動作確認

- Node.js 用のモジュール（ライブラリ）のコピー
  - CampusSquare から「PHC_06_素材.zip」をダウンロードし、中にある**'node_modules'フォルダと'mokkin.mp3'を 6.1 フォルダ内にコピー**

    ※ 'node_modules'フォルダ内に、'socket.io'と'express'というモジュール（ライブラリ）が入っています。これらのモジュールは本来、外部サイトから自身でダウンロードしますが、授業ではダウンロード済のものを利用します。自身でのダウンロード／インストール方法は付録 7・8 を参照してください。

- Node.js でサーバ側プログラム（app.js）を起動
  - スタート → すべてのプログラム → Node.js → **Node.js command prompt**
  - プロンプト（>）に続けて順に以下のコマンドを入力して Enter（↲）

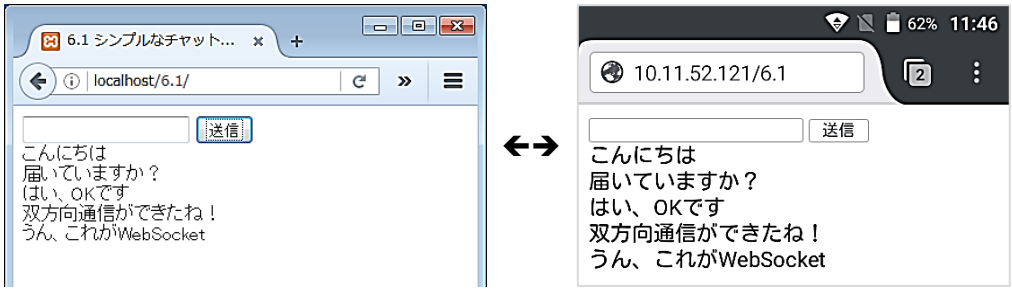
    >ipconfig↲　… サーバ PC の IPv4 アドレスをメモ（★）

    >cd 6.1↲　… C:￥Users￥（ユーザ名）￥6.1 フォルダに移動

    >node app.js↲　… Node.js を使って app.js を実行 → 何も表示されなければ起動 OK

- PC とスマホの Firefox を立ち上げて確認
  - メモしたアドレス（★）を入力して Enter　（例：**10.11.123.456**）

    （Node.js の http サーバを使う場合、6.1 などのディレクトリ名は指定しません）
  - テキストボックスに適当なメッセージを入力して 送信 ボタンを押すと、両方の画面に即座にメッセージが表示されれば OK（リアルタイムな双方向通信ができた）



PC 側　　スマホ側

- サーバ側プログラム（app.js）の止め方
  - Node.js command prompt で、キーボードから「**Ctrl + C**」を入力

Learn 解　説

⑩ **<script src='/socket.io/socket.io.js'></script>**

socket.io を使う時の定型文なので、悩まずこのまま書きましょう。

⑪ **const socket = io();**

クライアント側のブラウザをソケット通信サーバに接続します。ブラウザがこれを実行すると、サーバ側プログラム (app.js) の⑤で connection イベントが発生します。

⑫ **const data = { text: txt.value };**

ソケット通信では、単なる変数ではなく、オブジェクトを送受信します。{ }で囲むとオブジェクトが定義できます。{ }内の text: txt.value は、text という名前の属性 (プロパティ) にテキストボックスの文字列を設定する、という意味です。属性の名前は自由に定義できます。

⑬ **socket.emit('all', data);**

ソケット通信でデータを送っています。( )内は、(イベント名, 送信するオブジェクト)です。イベント名は自由に定義でき、イベント名によって処理を変えたりすることができます。

この場合は'all'というイベント名で、⑫で作った data オブジェクトを送信しています。ブラウザがこの行を実行すると、サーバ側プログラム (app.js) の⑥および⑦が実行されます。

⑭ **socket.on('msg', function(data) { receiveData(data); });**

app.js の⑥および⑦によってデータが送られてくると、 socket.on が実行されます。

この場合、送られてきた socket が'msg'というイベント名だったら function(data) {...} 内の処理を実行します。実際には receiveData 関数を実行しています

⑮ **const newMsg = document.createElement('div');**

document.createElement()は新しい HTML の要素を作ります。( )内は HTML の要素名です。

ここでは新しい div 要素を作っています。なお、この段階ではまだブラウザに表示されません。

⑯ **msgArea.appendChild(newMsg);**

親要素.appendChild(子要素)は、親となる HTML 要素の中に入れ子にして子要素を追加します。

この場合、親要素は msgArea という div 要素です。この msgArea 中に⑮で作った newMsg という要素を入れ子にして追加しています。つまり、メッセージが来たら、newMsg を作って、newMsg にメッセージ文字列を入れて、msgArea 内に追加する、という意味です。

⑰ **const sound = new Audio(); 以降**

着信音を鳴らす処理です。まず音声ファイルを扱う Audio オブジェクトを作り、src 属性にファイル名を指定して、play()メソッドで音声ファイルを再生しています。

✈ 6.2 おしゃべりなチャット（Chrome 限）

	PC	And	iOS
Firefox	△	×	×
Chrome	○	○	○

6.1 のチャットアプリを改造して、音声合成でメッセージを読み上げる機能を付けます。

Input クライアント側プログラム（HTML・JavaScript）を入力

- **C:￥Users￥（ユーザ名）￥6.1 をフォルダごと複製**して**フォルダ名を 6.2 に**します。
- index.html を Brackets で開き、以下のコードを入力しましょう。**変更は網掛けの部分**です。
- **app.js（サーバ側プログラム）はなにも変更しません。**

※ ①や②などの丸数字は打ち込まないこと
　コメント文は打ち込み必須ではありません

C:￥Users￥（ユーザ名）￥**6.2**￥ **index.html**

```
（略）
        <title>6.2 おしゃべりなチャット（Chrome 限）</title>
    </head>
    <body>
            :
        （中略）
            :
        <script>
            const socket = io();            // 双方向通信用のサーバに接続

            // サーバへのデータ送信
                :
            （中略・送信処理は 6.1 と全く同じです）
                :

            // サーバからのデータ受信
            socket.on('msg', function(data) {   // 'msg'というイベントの socket が来たら
                receiveData(data);              // receiveData を実行
            });
            function receiveData(data) {        // データの受信処理
                const newMsg = document.createElement('div'); // 新規メッセージ要素作成
                newMsg.innerHTML = data.text;   // newMsg に data の text プロパティを入れる
                const msgArea = document.getElementById('msgArea');   // 表示エリアを取得
                msgArea.appendChild(newMsg);    // 表示エリアに子要素として newMsg を追加
                const speech = new SpeechSynthesisUtterance();  // 音声合成オブジェクト
                speech.lang = 'Ja-JP';                  // 言語を日本語に設定
                speech.text = data.text;                // 話す文字列の設定
                window.speechSynthesis.speak(speech);   // 音声合成の実行                ①
            }
        </script>
（以下略）
```

Check スマホ 2 台で動作確認

- Node.js でサーバ側プログラム（app.js）を起動
  - Node.js command prompt の現在位置が C:￥Users￥（ユーザ名）￥6.1 であることを想定
  - プロンプトに続けて順に以下のコマンドを入力して Enter（↲）

    >cd..↲　　… ひとつ上のフォルダに移動（C:￥Users￥（ユーザ名）に移動）

    >cd 6.2↲　　… C:￥Users￥（ユーザ名）￥6.2 フォルダに移動

    >node app.js↲　　… Node.js を使って app.js を実行 → 何も表示されなければ起動 OK

- 2 台のスマホの Chrome で確認（隣の人と、または教員と）
  - アドレス例：**10.11.123.456**
  - テキストボックスに適当なメッセージを入力して 送信 ボタンを押すと、両方の画面に即座にメッセージが表示され、さらに日本語で音声が流れれば OK

- サーバ側プログラム（app.js）の止め方
  - Node.js command prompt で、キーボードから「**Ctrl + C**」を入力

Learn 解　説

① **`window.speechSynthesis.speak(speech);` など**

音声合成の方法の詳細は 3.7 を参照しましょう。

✈ 6.3 ぶるぶるチャット

	PC	And	iOS
Firefox	×	○	○
Chrome	×	○	○

テキストメッセージ以外のデータをリアルタイムに送るチャットアプリを作ります。

Input クライアント側プログラム (HTML・JavaScript) を入力

- **C:￥Users￥ (ユーザ名) ￥6.1 をフォルダごと複製**して**フォルダ名を 6.3 に**します。
- index.html を Brackets で開き、以下のコードを入力しましょう。**変更は網掛けの部分**です。
- **app.js (サーバ側プログラム) はなにも変更しません。**

※ ①や②などの丸数字は打ち込まないこと
　コメント文は打ち込み必須ではありません

C:￥Users￥ (ユーザ名) ￥**6.3**￥ **index.html**

```
( 略 )
        <title>6.3 ぶるぶるチャット</title>
    </head>
    <body>
        <!-- テキストボックスは削除しておきます-->
        <input type='button' id='btn' value='送信'>      <!-- 送信ボタン -->
        <div id='msgArea'></div>                         <!-- メッセージの表示エリア -->
        <script src='/socket.io/socket.io.js'></script>  <!-- socket.io の読み込み -->
        <script>
            const socket = io();            // 双方向通信用のサーバに接続
            // サーバへのデータ送信
            const btn = document.getElementById('btn');   // ボタンを取得
            btn.addEventListener('click', sendData);      // click で sendData を実行
            function sendData() {           // データの送信処理
                const data = {              // 送信するデータオブジェクトを作る
                    time: 1000              // time というプロパティに振動時間[ms]を設定 ①
                };
                socket.emit('others', data);    // 'others'というイベントで data 送信     ②
            }
            // サーバからのデータ受信
            socket.on('msg', function(data) {   // 'msg'というイベントの socket が来たら
                receiveData(data);              // receiveData を実行
            });
            function receiveData(data) {            // データの受信処理
                const msgArea = document.getElementById('msgArea');
                msgArea.innerHTML = 'こんにちは';   // msgArea に表示
                navigator.vibrate(data.time);       // 指定時間バイブレータを振動させる
                                    // 次のページに続きます
                window.setTimeout(function() {      // 指定時間後に何かをする setTimeout
                    msgArea.innerHTML = '';         // msgArea を空白に
                }, data.time);                      // data.time[ms]後に
            }
        </script>
( 以下略 )
```

Check スマホ 2 台で動作確認

- Node.js でサーバ側プログラム (app.js) を起動
  - Node.js command prompt の現在位置が C:￥Users￥ (ユーザ名) ￥6.2 であることを想定
  - プロンプトに続けて順に以下のコマンドを入力して Enter (↲)

    >cd..↲　　… ひとつ上のフォルダに移動 (C:￥Users￥ (ユーザ名) に移動)

    >cd 6.3↲　　… C:￥Users￥ (ユーザ名) ￥6.3 フォルダに移動

    >node app.js↲　　… Node.js を使って app.js を実行 → 何も表示されなければ起動 OK

- 2 台のスマホの **Firefox** で確認 (隣の人と、または教員と)
  - アドレス例： **10.11.123.456**
  - 送信 ボタンを押すと先方のバイブレータが振動し、1 秒後に振動とメッセージが消えれば OK

- サーバ側プログラム (app.js) の止め方
  - Node.js command prompt で、キーボードから「**Ctrl + C**」を入力

Learn 解　説

① **const data = { time: 1000 };**

送信するオブジェクト内の属性名は自由に設定できます。ここでは、time という属性に 1000 という数値 (バイブレータを振動させる時間) を設定しています。

② **socket.emit('others', data);**

サーバに data オブジェクトを送る命令です。6.2 までは'all'イベントを付けていましたが、ここでは'others'イベントを付けています。これにより、app.js (サーバ側プログラム) の⑧および⑨が実行され、自分以外の全員に送るようにしています。

③ **navigator.vibrate(data.time);**

4.1c でも使った、スマホのバイブレータを振動させる navigator.vibrate(時間[ms])です。

④ **window.setTimeout(function() { ... }, data.time);**

指定した時間 (遅延時間) 後に 1 回だけなんらかの処理を実行させる window.setTimeout()です。( )内は、(処理，遅延時間)です。この場合は、data.time(①で指定)が過ぎたら、function() { ... } の中の処理を実行しています。

✈ 6.4 画像付きチャット

	PC	And	iOS
Firefox	○	○	○
Chrome	○	○	○

テキストメッセージと画像をあわせて送るチャットアプリを作りながら、複数のメディアをやりとりする方法を学びます。

Input クライアント側プログラム (HTML・JavaScript) を入力

- **C:￥Users￥ (ユーザ名) ￥6.1 をフォルダごと複製**して**フォルダ名を 6.4 に**します。
- index.html を Brackets で開き、以下のコードを入力しましょう。**変更は網掛けの部分**です。
- **app.js (サーバ側プログラム) はなにも変更しません。**

※ ①や②などの丸数字は打ち込まないこと
　コメント文は打ち込み必須ではありません

C:￥Users￥ (ユーザ名) ￥6.4￥ index.html

```
( 略 )
        <title>6.4 画像付きチャット</title>
    </head>
    <body>
        <input type='text' id='txt'>                      <!-- テキストボックス -->
        <input type='file' accept='image/*' id='imgChooser'> <!-- 画像選択 -->          ①
        <canvas width='64' height='64' id='img'></canvas>    <!-- 画像を置く canvas -->
        <input type='button' id='btn' value='送信'>       <!-- 送信ボタン -->
        <div id='msgArea'></div>                          <!-- メッセージの表示エリア -->
        <script src='/socket.io/socket.io.js'></script>  <!-- socket.io の読み込み -->

        <script>
            const socket = io();              // 双方向通信用のサーバに接続
            const canvas = document.getElementById('img');
            const context = canvas.getContext('2d');
            // 画像の選択と canvas への読み込み ( 3.3 を参照 )                          ①
            const imgChooser = document.getElementById('imgChooser');
            const fileReader = new FileReader();
            const img = new Image();
            // 画像を選ぶ・撮る
            imgChooser.addEventListener('change', readImage);
            function readImage() {
                const file = imgChooser.files[0];
                fileReader.readAsDataURL(file);
            }
            // 画像を読み込む
            fileReader.addEventListener('load', loadImage);
            function loadImage() {
                img.src = fileReader.result;
            }
```

// 次のページに続きます

```
            // 読み込んだ画像を canvas に表示する
            img.addEventListener('load', drawImage);
            function drawImage() {
                context.drawImage(img, 0, 0, 64, 64);
            }

            // サーバへのデータ送信
            const btn = document.getElementById('btn');       // ボタンを取得
            btn.addEventListener('click', sendData);          // click で sendData を実行
            function sendData() {               // データの送信処理
                const txt = document.getElementById('txt');   // テキストボックスを取得
                if (txt.value == '') {          // テキストボックスが空なら
                    return;                     // 何もしないで抜ける
                }
                const data = {                  // 送信するデータオブジェクトを作る      ②
                    text: txt.value,            // text というプロパティにテキストを入れる
                    pict: canvas.toDataURL()    // pict というプロパティに画像を入れる
                };
                socket.emit('all', data);       // 'all'というイベントで data 送信
                txt.value = '';                 // テキストボックスを空にする
            }

            // サーバからのデータ受信
            socket.on('msg', function(data) {   // 'msg'というイベント名の socket が来たら
                receiveData(data);              // receiveData を実行
            });
            function receiveData(data) {        // データの受信処理
                const newMsg = document.createElement('div'); // 新規メッセージ要素の作成
                newMsg.innerHTML = data.text;   // newMsg に data の text プロパティを入れる
                const msgArea = document.getElementById('msgArea');   // 表示エリアを取得
                msgArea.appendChild(newMsg);    // 表示エリアに子要素として newMsg を追加
                const newImg = document.createElement('img'); // 画像表示用 img 要素作成 ③
                newImg.width = 64;                            // 画像の幅
                newImg.height = 64;                           // 画像の高さ
                newMsg.appendChild(newImg);       // newMsg の子要素として newImg を追加  ④
                newImg.src = data.pict;                       // img の src に画像を設定
            }
        </script>
( 以下略 )
```

Check PC とスマホ（または 2 台のスマホ）で動作確認

- Node.js でサーバ側プログラム（app.js）を起動
  - Node.js command prompt の現在位置が C:￥Users￥（ユーザ名）￥6.3 であることを想定
  - プロンプトに続けて順に以下のコマンドを入力して Enter（↲）

    >**cd..**↲　　… ひとつ上のフォルダに移動（C:￥Users￥（ユーザ名）に移動）

    >**cd 6.4**↲　　… C:￥Users￥（ユーザ名）￥6.4 フォルダに移動

    >**node app.js**↲　　… Node.js を使って app.js を実行 → 何も表示されなければ起動 OK

- PC とスマホの Firefox を立ち上げて確認
  - アドレス例：**10.11.123.456**
  - テキストボックスに適当なメッセージを入力
  - Browse... または 参照... ボタンを押してカメラまたはスマホ内の画像を選ぶ
  - 送信 ボタンを押したら、両方の画面に即座にメッセージと画像が表示されれば OK

スマホ側　　PC 側

- サーバ側プログラム（app.js）の止め方
  - Node.js command prompt で、キーボードから「**Ctrl + C**」を入力

Learn 解　説

① **`<input type='file' accept='image/*' id='imgChooser'>` など**

3.3 で用いた、スマホの標準カメラアプリで写真を撮ったり、カメラロールの写真を選んで表示する処理です。詳しくは 3.3 を参照してましょう。

② **`const data = { text: text.value, pict: canvas.toDataURL() };`**

オブジェクトには複数の属性を設定できます。ここでは text 属性と pict 属性の二つを、カンマで区切って設定しています。pict 属性には画像を設定しています。といっても、画像データ(img.src など) を直接設定することはできません。そこで、canvas.`toDataURL()`を使って、画像を Data URL (base64) 形式という特殊なテキスト文字列に変換しています。こうすることで、socket 通信で画像を送り合うことができます。

③ **`const newImg = document.createElement('img');`**

6.1～6.3 では div 要素を新しく作ってメッセージ文字列を追加していきました。ここでは同様に img 要素を新しく作っています。

④ **`newMsg.appendChild(newImg);`**

newMsg は msgArea の子要素ですが、さらに newMsg の子要素として newImg を追加しています。これで、メッセージの横に画像が表示されるようになります。

✈ 6.5 Face to Face (Firefox 限)

	PC	And	iOS
Firefox	○	○	○
Chrome	△	×	×

カメラの映像 (動画) を送りあうビデオチャットを作りながら、ビデオ通話や遠隔監視の方法を学びます。

Input クライアント側プログラム (HTML・JavaScript) を入力

- **C:￥Users￥ (ユーザ名) ￥6.1 をフォルダごと複製**して**フォルダ名を 6.5 に**します。
- index.html を Brackets で開き、以下のコードを入力しましょう。変更は網掛けの部分です。
- **app.js (サーバ側プログラム) はなにも変更しません。**

※ ①や②などの丸数字は打ち込まないこと
　コメント文は打ち込み必須ではありません

C:￥Users￥ (ユーザ名) ￥**6.5**￥ **index.html**

```
( 略 )
        <title>6.5 Face to Face ( Firefox 限 )</title>
    </head>
    <body>
        <!-- 自分のカメラ映像を表示する video 要素 ( 3.6 を参照 ) -->                    ①
        <video width='144' height='192' autoplay id='video'></video>
        <!-- 自分のカメラ映像をキャプチャする canvas 要素 ( 非表示にしておく ) -->
        <canvas width='144' height='192' hidden='true' id='local'></canvas>
        <!-- 相手のカメラ映像を表示する img 要素 -->
        <img width='144' height='192' id='remote'>

        <script src='/socket.io/socket.io.js'></script>  <!-- socket.io の読み込み -->

        <script>
            const socket = io();                // 双方向通信用のサーバに接続

            //カメラを使う ( 3.6 を参照 )
            const media = navigator.mediaDevices.getUserMedia({
                    video: true,    // カメラを使うか否か
                    audio: false    // マイクを使うか否か
            });
            media.then(onSuccess, onError);     // ストリーム取得成功・失敗時の関数
            const video = document.getElementById('video');     // video 要素を取得
            function onSuccess(stream) {        // 成功時の onSuccess 関数
                video.src = window.URL.createObjectURL(stream); // video 要素で stream 表示
                video.addEventListener('timeupdate', sendData); // 更新時 sendData 実行  ②
            }
            function onError(err) {             // エラー時の onError 関数
                window.alert(err);              // 警告ダイアログに err を表示
            }
                                    // 次のページに続きます
```

```
            // サーバへのデータ送信
            function sendData() {
                // 画像を canvas にキャプチャ（3.6 を参照）
                const canvas = document.getElementById('local');
                const context = canvas.getContext('2d');
                context.drawImage(video, 0, 0, 144, 192);
                const data = {                  // 送信するデータオブジェクトを作る      ③
                    pict: canvas.toDataURL()    // pict というプロパティに画像を入れる
                };
                socket.emit('others', data);    // 'others'というイベントで data を送信  ④
            }

            // サーバからのデータ受信
            socket.on('msg', function(data) {   // 'msg'というイベント名の socket が来たら
                receiveData (data);             // receiveData を実行
            });
            function receiveData (data) {       // データの受信処理
                const remote = document.getElementById('remote'); // 表示する img 要素
                remote.src = data.pict;                           // src 属性に画像を設定
            }
        </script>
（以下略）
```

Check PC とスマホ（または 2 台のスマホ）で動作確認

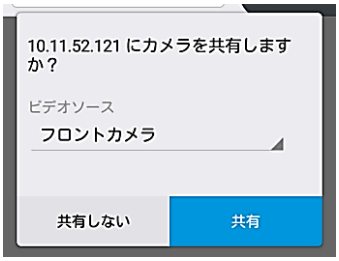
- Node.js でサーバ側プログラム（app.js）を起動
  - Node.js command prompt の現在位置が C:￥Users￥（ユーザ名）￥6.4 であることを想定
  - プロンプトに続けて順に以下のコマンドを入力して Enter（↲）

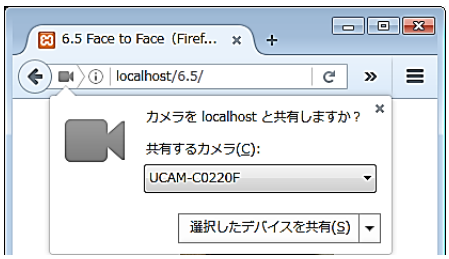
    >cd..↲　　… ひとつ上のフォルダに移動（C:￥Users￥（ユーザ名）に移動）

    >cd 6.5↲　　… C:￥Users￥（ユーザ名）￥6.5 フォルダに移動

    >node app.js↲　　… Node.js を使って app.js を実行

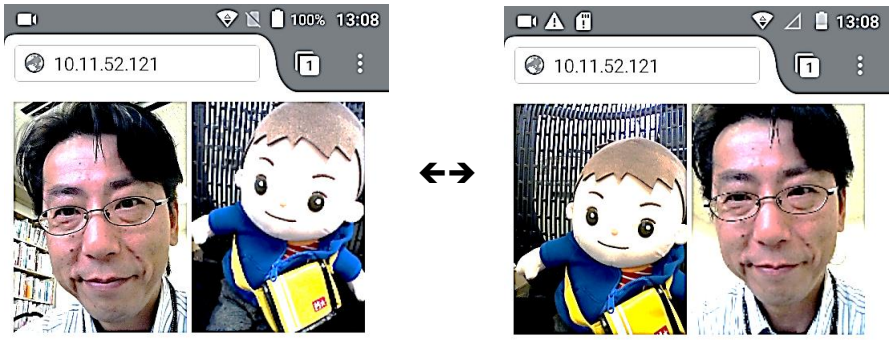
- PC※1 とスマホの **Firefox** を立ち上げて確認
  - アドレス例：**10.11.123.456**
  - カメラを共有するか否かの確認が出るので、使うカメラを選んで 共有 をタップ





※1 PC の場合、内蔵の Web カメラまたは外付けの Web カメラが必要です。

- 画面の左側に自分のカメラの映像、右側に相手のカメラの映像が表示されれば OK



スマホ 1（または PC1）の画面　　スマホ 2（または PC2）の画面

- サーバ側プログラム（app.js）の止め方
  - Node.js command prompt で、キーボードから「**Ctrl + C**」を入力

Learn 解　説

① **<!-- 自分のカメラ映像を表示する video 要素（3.6 を参照） -->** など

3.6 で用いた、標準カメラアプリを使わずに、カメラデバイスに直接アクセスして映像を取り込む方法です。詳しくは 3.6 を参照してましょう。画面の左半分に置いた video 要素にカメラの映像を表示しています。

② **video.addEventListener('timeupdate', sendData);**

video 要素は、その映像が更新された時に timeupdate イベントを発生させます。これを使い、映像が更新されたら sendData を実行し、映像をキャプチャして socket 通信で送るようにしています。

③ **const data = { pict: canvas.toDataURL() };**

カメラデバイスの映像（ストリーム）をいったん canvas にキャプチャしたら、あとは 6.4 と同様に、data オブジェクトの pict 属性に Data URL（base64）形式に変換した画像を持たせます。

④ **socket.emit('others', data);**

6.3 と同様に、自分以外の全員に対してデータを送信しています。

データを持ち帰る

- 本日作成したデータは全て USB メモリなどにコピーして持ち帰りましょう
  - C:￥xampp￥htdocs 内の以下のフォルダ (5 個)
  - 6.1　6.2　6.3　6.4　6.5

本日の課題

① 6.1、6.5 のコードを完成させる

・ 資料のコードをもとに自分なりのアレンジを加えても構いません。その際は②の readme.txt の所感にアレンジ内容を書いてください。効果的なアレンジであれば成績評価に加点します。

② readme.txt というテキストファイルを作り、以下を書く

・ 学籍番号 と 氏名
・ 所感　(考えたこと、感想、応用アイデアなど。字数不問だが数行は書いてほしい。)
・ 質問　(無ければ不要)

【提出方法】

・ ①のフォルダ 2 個と、②の「readme.txt」1 個、全てをまとめて zip 圧縮
・ zip のファイル名は「学籍番号.zip」とする　(例：12345nhu.zip)
・ CampusSquare のレポート「フィジカルコンピューティング **06**」から提出

【提出締切】　11 月 17 日 (木)　15:00　(遅れてしまった場合は担当教員に相談のこと)