



# 16. Socket.io とは

## WebSocket とは

WebSocket は Web でリアルタイムで双方向通信するプロトコルです。 `ws:` や `wss:` などのURI スキームでクライアントとサーバでデータ通信します。

## ポーリングとは

ポーリング は、クライアントが一定間隔サーバに HTTPリクエストする方法です。例えばチャットが書き込まれたかサーバに数秒後に確認することで、リアルタイム性を実現させます。また、サーバからクライアントに擬似的にプッシュする ロングポーリング (Comet) 方式もあります。

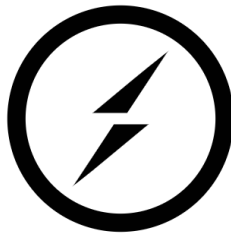
## HTTP のメリット

Ajax のようにHTTP はリクエスト&レスポンスするたびに新しいコネクションを確立するため、比較的速度が遅くなる傾向にあります。 WebSocket を利用すると低コスト通信で、Ajax よりもリアルタイム性を実現しやすくなります。

## Socket.io とは

Socket.io は、Node.js で WebSocket を利用するモジュールです。

[Socket.io 公式](#) 



# Socket.IO

Bidirectional and low-latency communication for every platform

Get started

Documentation

WebSocket だけでなく、Ajax ロングポーリング をはじめとする複雑な通信にも対応しています。また、クライアントアプリの機能もあり、プログラムしやすいのも特徴です。

## サーバ

Socket.io では クライアントとサーバ両方のプログラムを開発 できます。まずサーバ側の基本機能です。

## インストール

Socket.io を利用するには、npm で socket.io をインストールします。

```
% npm i socket.io
```

## on イベント

Socket.io の通信には、クライアントとサーバの接続を確立する必要があります。on() イベントを利用すると、接続、切断をはじめとするイベントを登録できます。

## io を直接利用する場合

`on()` にイベント名を指定して、コールバック関数でデータを取得します

```
io.on(イベント名, (データ) => {  
  //処理  
})
```

## socket を利用する場合

```
socket.on(イベント名, (データ) => {  
  //処理  
})
```

## connection イベント

---

`connection` イベントは Socketが接続確立すると呼ばれます。 コールバック関数の引数には、 `socket` オブジェクトが利用できます。

```
io.on('connection', (socket) => {  
  //処理  
})
```

## disconnect イベント

---

`disconnect` イベントは Socketが切断すると呼ばれます。

```
socket.on('disconnect', () => {  
  //処理  
})
```

## ルームを利用する場合

---

チャットのようなルームを指定して、送信先を振り分けることもできます。

```
io.of('/').in(ルーム名).clients(err, clients => {  
  //処理  
})
```

# データ送信 emit()

サーバからクライアントにデータ送信するには、`emit()` メソッドを利用します。データ送信の方法はいくつかあるので、状況によって使い分けます。

## すべてのクライアントに送信

`io` で `emit()` すると、すべてのクライアントに送信します。

```
io.emit(イベント名, データ);
```

## 送信者のみに送信

`socket` で `emit()` すると、送信者のみに送信します。

```
socket.emit(イベント名, データ);
```

## ブロードキャスト送信

ブロードキャスト送信は、送信者以外のすべてのクライアント送信します。

```
io.broadcast.emit(イベント名, データ);  
//または  
socket.broadcast.emit(イベント名, データ);
```

## 特定のクライアントに送信

`Socket ID` を指定して、特定のクライアントに送信します。

```
io.to(SocketID).emit(イベント名, データ);
```

## 特定のルームに送信

`io.to()` で、ルーム（グループ）ごとに送信します。

```
io.to(ルーム名).emit(イベント名, データ);
```

# クライアント

クライアントもサーバと同じような記述で **Socket.io** を利用できます。 **node\_modules** にインストールされた **socket.io.js** を利用します。

## Socket.io クライアントのインストール

---

**head** タグで **socket.io.js** をインストールします。

```
<script type="text/javascript" src="/socket.io/socket.io.js"></script>
```

## connection イベント

---

サーバ同様に **connection** イベントが利用できます。 **socket.id** や **socket.connected** で接続の ID や状況が確認できます。

```
socket.on('connect', () => {  
  console.log(socket.id);  
  console.log(socket.connected);  
});
```

## disconnect イベント

---

サーバ同様に **disconnect** イベントが利用できます。

```
socket.on('disconnect', () => {  
  //処理  
})
```

## サーバに送信

---

**socket.emit()** メソッドでクライアントからサーバの送信します。イベント名は、サーバの設定にあわせます。

```
socket.emit(イベント名, データ);
```

## サーバから受信

---

`socket.on()` で接続した `socket` に対するイベントを登録します。イベント名はサーバとあわせておきます。

```
socket.on(イベント名, (data) => {  
    //処理  
})
```

当サイトの教材をはじめとするコンテンツ（テキスト、画像等）の無断転載・無断使用を固く禁じます。これらのコンテンツについて権利者の許可なく複製、転用等する事は法律で禁止されています。尚、当ウェブサイトの内容をWeb、雑誌、書籍等へ転載、掲載する場合は「ロジコヤ」までご連絡ください。