SOCKET.IO クイックリファレンス

イベント、ルーム、ネームスペース、ミドルウェア、リアルタイムパターン

セットアップ

サーバーセットアップ（Node.js）

```
import { Server } from "socket.io";
const io = new Server(3000, {
  cors: { origin: "http://localhost:5173" }
});
```

クライアントセットアップ

```
import { io } from "socket.io-client";
const socket = io("http://localhost:3000");
```

Express との組み合わせ

```
import express from "express";
import { createServer } from "http";
import { Server } from "socket.io";
const app = express();
const server = createServer(app);
const io = new Server(server);
```

サーバーオプション

`cors`	クロスオリジン用の CORS 設定
`path`	カスタムパス（デフォルト: /socket.io）
`pingInterval`	ハートビート間隔（ミリ秒、デフォルト 25000）
`pingTimeout`	切断までのタイムアウト（デフォルト 20000）
`maxHttpBufferSize`	最大メッセージサイズ（バイト、デフォルト 1MB）

イベント

組み込みイベント（サーバー）

`connection`	クライアントが接続した
`disconnect`	クライアントが切断した
`disconnecting`	クライアントが切断中（まだルームに参加中）
`error`	エラーイベント

組み込みイベント（クライアント）

`connect`	サーバーに接続した
`disconnect`	サーバーから切断した
`connect_error`	接続に失敗した
`reconnect`	再接続に成功した
`reconnect_attempt`	再接続を試みている

接続のライフサイクル

```
io.on("connection", (socket) => {
  console.log(`connected: ${socket.id}`);
  socket.on("disconnect", (reason) => {
    console.log(`disconnected: ${reason}`);
  });
});
```

送信

サーバーからの送信

```
socket.emit("hello", { msg: "world" });
socket.emit("data", arg1, arg2);
io.emit("broadcast", data);
```

クライアントからの送信

```
socket.emit("chat:message", { text });
socket.emit("update", data, (res) => {
  console.log("ack:", res);
});
```

送信パターン

`socket.emit(ev, data)`	このソケットのみに送信
`io.emit(ev, data)`	接続中のすべてのクライアントに送信
`socket.broadcast.emit()`	送信者以外の全クライアントに送信
`io.to(room).emit()`	ルーム内のすべてのクライアントに送信
`socket.to(room).emit()`	送信者以外のルームメンバーに送信

ブロードキャスト

ブロードキャストメソッド

```
io.emit("msg", data);
socket.broadcast.emit("msg", data);
io.to("room1").emit("msg", data);
io.except("room2").emit("msg", data);
```

揮発性と圧縮

`socket.volatile.emit()`	クライアントが準備できていない場合は破棄（バッファなし）
`socket.compress(true).emit()`	メッセージごとの圧縮を有効化
`io.local.emit()`	ローカルサーバーのみにブロードキャスト（マルチノード）
`socket.timeout(5000).emit()`	応答確認タイムアウト付きで送信

ルーム

ルーム操作

```
socket.join("room-1");
socket.join(["room-1", "room-2"]);
socket.leave("room-1");
io.to("room-1").emit("msg", data);
```

ルームのプロパティ

`socket.rooms`	このソケットが参加しているルームのセット
`socket.id`	各ソケットは自分の ID のルームに自動参加
`io.sockets.adapter.rooms`	すべてのルームとメンバーのマップ

ルームパターン

```
socket.on("join:room", (room) => {
  socket.join(room);
  io.to(room).emit("user:joined", socket.id);
});
socket.on("disconnecting", () => {
  for (const room of socket.rooms) {
    socket.to(room).emit("user:left", socket.id);
  }
});
```

ネームスペース

ネームスペースの作成

```
const chat = io.of("/chat");
const admin = io.of("/admin");
chat.on("connection", (socket) => {
  chat.emit("user:online", socket.id);
});
```

ネームスペースへのクライアント接続

```
const chat = io("http://localhost:3000/chat");
const admin = io("http://localhost:3000/admin");
```

動的ネームスペース

```
io.of(/^\/project-\d+$/).on("connection",
  (socket) => {
    const ns = socket.nsp.name;
    console.log(`joined namespace: ${ns}`);
  }
);
```

ミドルウェア

サーバーミドルウェア

```
io.use((socket, next) => {
  const token = socket.handshake.auth.token;
  if (isValid(token)) return next();
  next(new Error("authentication failed"));
});
```

ネームスペースミドルウェア

```
const admin = io.of("/admin");
admin.use((socket, next) => {
  if (socket.handshake.auth.role === "admin")
    return next();
  next(new Error("not authorized"));
});
```

ミドルウェアのプロパティ

`socket.handshake.auth`	クライアントから送信された認証データ
`socket.handshake.headers`	初期リクエストの HTTP ヘッダー
`socket.handshake.query`	接続 URL のクエリパラメータ
`socket.data`	ミドルウェアで付与した任意データ

エラーハンドリング

サーバー側のエラー

```
socket.on("action", (data, callback) => {
  try {
    const result = process(data);
    callback({ status: "ok", data: result });
  } catch (err) {
    callback({ status: "error", msg: err.message });
  }
});
```

クライアント側のエラー

```
socket.on("connect_error", (err) => {
  console.log("connection error:", err.message);
});
socket.io.on("reconnect_failed", () => {
  console.log("reconnection failed");
});
```

クライアントの再接続オプション

`reconnection`	自動再接続を有効化（デフォルト true）
`reconnectionAttempts`	最大試行回数（デフォルト Infinity）
`reconnectionDelay`	初回遅延ミリ秒（デフォルト 1000）
`reconnectionDelayMax`	最大遅延ミリ秒（デフォルト 5000）

確認応答

クライアント送信、サーバーが確認

```
// クライアント
socket.emit("save", data, (response) => {
  console.log("server ack:", response);
});
// サーバー
socket.on("save", (data, callback) => {
  callback({ saved: true, id: 42 });
});
```

サーバー送信、クライアントが確認

```
// サーバー
socket.emit("ping", (response) => {
  console.log("client ack:", response);
});
// クライアント
socket.on("ping", (callback) => {
  callback("pong");
});
```

タイムアウト付き

```
socket.timeout(5000).emit("save", data,
  (err, response) => {
    if (err) console.log("timeout!");
    else console.log("ack:", response);
  }
);
```

よく使うパターン

チャットルーム

```
io.on("connection", (socket) => {
  socket.on("chat:join", (room) => {
    socket.join(room);
    socket.to(room).emit("chat:joined",
      socket.id);
  });
  socket.on("chat:message", ({ room, text }) => {
    io.to(room).emit("chat:message", {
      from: socket.id, text
    });
  });
});
```

オンラインプレゼンス

```
const users = new Map();
io.on("connection", (socket) => {
  users.set(socket.id, socket.handshake.auth);
  io.emit("users:list", [...users.values()]);
  socket.on("disconnect", () => {
    users.delete(socket.id);
    io.emit("users:list", [...users.values()]);
  });
});
```

レートリミット

```
io.use((socket, next) => {
  const ip = socket.handshake.address;
  if (rateLimiter.consume(ip)) return next();
  next(new Error("rate limit exceeded"));
});
```