

Référence rapide Socket.IO

Événements, salles, espaces de noms, middleware, modèles temps réel

Configuration

Configuration serveur (Node.js)

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

Configuration client

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

Avec Express

```
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);
```

Options du serveur

cors	Configuration CORS pour les requêtes cross-origin
path	Chemin personnalisé (défaut : /socket.io)
pingInterval	Intervalle de battement de cœur (ms, défaut 25000)
pingTimeout	Délai avant déconnexion (défaut 20000)
maxHttpBufferSize	Taille maximale du message en octets (défaut 1 Mo)

Événements

Événements intégrés (serveur)

connection	Un client se connecte
disconnect	Un client se déconnecte
disconnecting	Le client est en cours de déconnexion (encore dans les salles)
error	Événement d'erreur

Événements intégrés (client)

connect	Connecté au serveur
disconnect	Déconnecté du serveur
connect_error	Échec de connexion
reconnect	Reconnexion réussie
reconnect_attempt	Tentative de reconnexion en cours

Cycle de vie de la connexion

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

Émission

Émission serveur

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

Émission client

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

Modèles d'émission

socket.emit(ev, data)	Envoyer à ce socket uniquement
io.emit(ev, data)	Envoyer à tous les clients connectés
socket.broadcast.emit()	Tous les clients sauf l'émetteur
io.to(room).emit()	Tous les clients d'une salle
socket.to(room).emit()	Membres de la salle sauf l'émetteur

Diffusion

Méthodes de diffusion

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

Volatile et compression

socket.volatile.emit()	Ignorer si le client n'est pas prêt (sans mise en tampon)
socket.compress(true).emit() io.local.emit()	Activer la compression par message Diffuser au serveur local uniquement (multi-nœuds)
socket.timeout(5000).emit()	Émettre avec délai d'attente pour l'accusé de réception

Salles

Opérations sur les salles

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

Propriétés des salles

socket.rooms	Ensemble des salles où se trouve ce socket
socket.id	Chaque socket rejoint automatiquement sa propre salle d'identifiant
io.sockets.adapter.rooms	Carte de toutes les salles et de leurs membres

Modèles de salles

```
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);
  }
});
```

Espaces de noms

Création d'espaces de noms

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

Connexion client à un espace de noms

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

Espaces de noms dynamiques

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

Middleware

Middleware serveur

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

Middleware d'espace de noms

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

Propriétés du middleware

socket.handshake.auth	Données d'authentification envoyées par le client
socket.handshake.headers socket.handshake.query	En-têtes HTTP de la requête initiale Paramètres de requête de l'URL de connexion
socket.data	Données arbitraires attachées dans le middleware

Gestion des erreurs

Erreurs côté serveur

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

Erreurs côté client

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

Options de reconnexion client

reconnection	Activer la reconnexion automatique (défaut : true)
reconnectionAttempts	Nombre max de tentatives (défaut : Infinity)
reconnectionDelay	Délai initial en ms (défaut : 1000)
reconnectionDelayMax	Délai maximal en ms (défaut : 5000)

Référence rapide Socket.IO

Accusés de réception

Client envoie, serveur accuse réception

```
// client
socket.emit("save", data, (response) => {
  console.log("server ack:", response);
});
// server
socket.on("save", (data, callback) => {
  callback({ saved: true, id: 42 });
});
```

Serveur envoie, client accuse réception

```
// server
socket.emit("ping", (response) => {
  console.log("client ack:", response);
});
// client
socket.on("ping", (callback) => {
  callback("pong");
});
```

Avec délai d'attente

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

Modèles courants

Salle de chat

```
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
    });
  });
});
```

Présence en ligne

```
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()]);
  });
});
```

Limitation de débit

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