Tham Khảo Nhanh Socket.IO

Events, rooms, namespaces, middleware, real-time patterns

Cài đặt

Cài đặt Server (Node.js)

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

Cài đặt Client

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

Dùng với 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);
```

Tùy chọn Server

cors	Cấu hình CORS cho cross-origin
path	Đường dẫn tùy chỉnh (mặc định: /socket.io)
pingInterval	Khoảng thời gian heartbeat (ms, mặc định 25000)
pingTimeout	Timeout trước khi ngắt kết nối (mặc định 20000)
maxHttpBufferSize	Kích thước tin nhắn tối đa tính bằng bytes (mặc định 1MB)

Events

Events tích hợp (Server)

connection	Client kết nối
disconnect	Client ngắt kết nối
disconnecting	Client đang ngắt kết nối (vẫn còn trong rooms)
error	Sự kiện lỗi

Events tích hợp (Client)

connect	Đã kết nối tới server
disconnect	Đã ngắt kết nối khỏi server
connect_error	Kết nối thất bại
reconnect	Kết nối lại thành công
reconnect_attempt	Đang thử kết nối lại

Vòng đời kết nối

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

Emitting

Server Emit

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

Client Emit

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

Các kiểu Emit

socket.emit(ev, data)	Gửi chỉ tới socket này
io.emit(ev, data)	Gửi tới tất cả clients đã kết nối
socket.broadcast.emit()	Tất cả clients trừ người gửi
io.to(room).emit()	Tất cả clients trong room
socket.to(room).emit()	Thành viên room trừ người gửi

Broadcasting

Các phương thức Broadcast

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

Volatile & Compressed

socket.volatile.emit()	Bỏ qua nếu client chưa sẵn sàng (không buffer)
socket.compress(true).emit()	Bật nén theo từng tin nhắn
io.local.emit()	Broadcast chỉ tới server local (multi-node)
socket.timeout(5000).emit()	Emit với timeout cho acknowledgement

Rooms

Thao tác Room

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

Thuộc tính Room

socket.rooms	Tập hợp các rooms socket đang tham gia
socket.id	Mỗi socket tự động tham gia room ID của nó
io.sockets.adapter.rooms	Map của tất cả rooms và thành viên

Mẫu Room

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

Namespaces

Tạo Namespaces

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

Client kết nối tới Namespace

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

Namespaces động

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

Middleware

Server Middleware

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

Namespace Middleware

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

Thuộc tính Middleware

socket.handshake.auth	Dữ liệu auth gửi từ client
socket.handshake.headers	HTTP headers từ request ban đầu
socket.handshake.query	Query parameters từ URL kết nối
socket.data	Dữ liệu tùy ý đính kèm trong middleware

Xử lý lỗi

Lỗi phía Server

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

Lỗi phía Client

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

Tùy chọn kết nối lại của Client

reconnection	Bật tự động kết nối lại (mặc định true)
reconnectionAttempts	Số lần thử tối đa (mặc định Infinity)
reconnectionDelay	Delay ban đầu tính bằng ms (mặc định 1000)
reconnectionDelayMax	Delay tối đa tính bằng ms (mặc định 5000)

Acknowledgements

Client Gửi, Server Xác nhận

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

Tham Khảo Nhanh Socket.IO

Server Gửi, Client Xác nhận

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

Với Timeout

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

Các mẫu thường dùng

Phòng 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
    });
  });
});
```

Trạng thái trực tuyến

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

Giới hạn tốc độ

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