REFERÊNCIA RÁPIDA DO SOCKET.IO

Eventos, salas, namespaces, middleware, padrões em tempo real

Configuração

Configuração do Servidor (Node.js)

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

Configuração do Cliente

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

Com 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);
```

Opções do Servidor

`cors`	Configuração CORS para origens cruzadas
`path`	Caminho personalizado (padrão: /socket.io)
`pingInterval`	Intervalo de heartbeat (ms, padrão 25000)
`pingTimeout`	Tempo limite antes de desconectar (padrão 20000)
`maxHttpBufferSize`	Tamanho máximo de mensagem em bytes (padrão 1MB)

Eventos

Eventos Internos (Servidor)

`connection`	Cliente conecta
`disconnect`	Cliente desconecta
`disconnecting`	Cliente está desconectando (ainda nas salas)
`error`	Evento de erro

Eventos Internos (Cliente)

`connect`	Conectado ao servidor
`disconnect`	Desconectado do servidor
`connect_error`	Conexão falhou
`reconnect`	Reconectado com sucesso
`reconnect_attempt`	Tentando reconectar

Ciclo de Vida da Conexão

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

Emissão

Emissão pelo Servidor

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

Emissão pelo Cliente

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

Padrões de Emissão

`socket.emit(ev, data)`	Envia apenas para este socket
`io.emit(ev, data)`	Envia para todos os clientes conectados
`socket.broadcast.emit()`	Todos os clientes exceto o remetente
`io.to(sala).emit()`	Todos os clientes na sala
`socket.to(sala).emit()`	Membros da sala exceto o remetente

Difusão

Métodos de Difusão

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

Volátil e Comprimido

`socket.volatile.emit()`	Descarta se o cliente não estiver pronto (sem buffer)
`socket.compress(true).emit()`	Habilita compressão por mensagem
`io.local.emit()`	Difunde apenas para o servidor local (multi-nó)
`socket.timeout(5000).emit()`	Emite com tempo limite para confirmação

Salas

Operações com Salas

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

Propriedades das Salas

`socket.rooms`	Conjunto de salas em que este socket está
`socket.id`	Cada socket entra automaticamente na sala de seu ID
`io.sockets.adapter.rooms`	Mapa de todas as salas e seus membros

Padrões com Salas

```
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

Criando Namespaces

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

Cliente Conectando a Namespace

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

Namespaces Dinâmicos

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

Middleware

Middleware do Servidor

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

Middleware de Namespace

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

Propriedades do Middleware

`socket.handshake.auth`	Dados de autenticação enviados pelo cliente
`socket.handshake.headers`	Cabeçalhos HTTP da requisição inicial
`socket.handshake.query`	Parâmetros de query da URL de conexão
`socket.data`	Dados arbitrários anexados no middleware

Tratamento de Erros

Erros no Lado do Servidor

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

Erros no Lado do Cliente

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

Opções de Reconexão do Cliente

`reconnection`	Habilita reconexão automática (padrão true)
`reconnectionAttempts`	Tentativas máximas (padrão Infinity)
`reconnectionDelay`	Atraso inicial em ms (padrão 1000)
`reconnectionDelayMax`	Atraso máximo em ms (padrão 5000)

Confirmações

Cliente Envia, Servidor Confirma

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

Servidor Envia, Cliente Confirma

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

Com Tempo Limite

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

Padrões Comuns

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

Presença Online

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

Limitação de Taxa

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