SOCKET.IO 快速参考

事件、房间、命名空间、中间件与实时模式

安装

服务端安装（Node.js）

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

客户端安装

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

与 Express 集成

```
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);
```

服务端选项

`cors`	跨域配置
`path`	自定义路径（默认：/socket.io）
`pingInterval`	心跳间隔（毫秒，默认 25000）
`pingTimeout`	断开连接前的超时（默认 20000）
`maxHttpBufferSize`	最大消息体积（字节，默认 1MB）

事件

内置事件（服务端）

`connection`	客户端连接
`disconnect`	客户端断开连接
`disconnecting`	客户端正在断开（仍在房间中）
`error`	错误事件

内置事件（客户端）

`connect`	已连接到服务器
`disconnect`	已与服务器断开连接
`connect_error`	连接失败
`reconnect`	重新连接成功
`reconnect_attempt`	正在尝试重新连接

连接生命周期

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

发送消息

服务端发送

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

客户端发送

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

发送模式

`socket.emit(ev, data)`	仅发送到此 socket
`io.emit(ev, data)`	发送给所有已连接的客户端
`socket.broadcast.emit()`	除发送者以外的所有客户端
`io.to(room).emit()`	房间内的所有客户端
`socket.to(room).emit()`	房间内除发送者以外的成员

广播

广播方法

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

Volatile 与压缩

`socket.volatile.emit()`	客户端未就绪时丢弃（不缓冲）
`socket.compress(true).emit()`	启用单条消息压缩
`io.local.emit()`	仅在本地服务器广播（多节点）
`socket.timeout(5000).emit()`	发送并等待确认，超时出错

房间

房间操作

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

房间属性

`socket.rooms`	当前 socket 所在的房间集合
`socket.id`	每个 socket 自动加入以自身 ID 命名的房间
`io.sockets.adapter.rooms`	所有房间及其成员的 Map

房间模式

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

命名空间

创建命名空间

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

客户端连接到命名空间

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

动态命名空间

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

中间件

服务端中间件

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

命名空间中间件

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

中间件属性

`socket.handshake.auth`	客户端发送的认证数据
`socket.handshake.headers`	初始请求的 HTTP 头
`socket.handshake.query`	连接 URL 中的查询参数
`socket.data`	在中间件中附加的任意数据

错误处理

服务端错误

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

客户端错误

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

客户端重连选项

`reconnection`	启用自动重连（默认 true）
`reconnectionAttempts`	最大尝试次数（默认无限）
`reconnectionDelay`	初始延迟毫秒（默认 1000）
`reconnectionDelayMax`	最大延迟毫秒（默认 5000）

确认回执

客户端发送，服务端确认

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

服务端发送，客户端确认

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

带超时

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

常用模式

聊天室

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

在线状态

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

限流

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