

Socket.IO 빠른 참조

이벤트, 룸, 네임스페이스, 미들웨어, 실시간 패턴

설정

서버 설정 (Node.js)

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

클라이언트 설정

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

Express와 함께 사용

```
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);
```

서버 옵션

cors	크로스 오리진을 위한 CORS 설정
path	커스텀 경로 (기본: /socket.io)
pingInterval	하트비트 간격 (ms, 기본 25000)
pingTimeout	연결 끊김 전 타임아웃 (기본 20000)
maxHttpBufferSize	최대 메시지 크기 바이트 (기본 1MB)

이벤트

내장 이벤트 (서버)

connection	클라이언트 연결
disconnect	클라이언트 연결 끊김
disconnecting	클라이언트 연결 끊기는 중 (아직 룸에 있음)
error	에러 이벤트

내장 이벤트 (클라이언트)

connect	서버에 연결됨
disconnect	서버에서 연결 끊김
connect_error	연결 실패
reconnect	재연결 성공
reconnect_attempt	재연결 시도 중

연결 생명주기

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

이벤트 발신

서버 발신

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

클라이언트 발신

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

발신 패턴

socket.emit(ev, data)	이 소켓에만 전송
io.emit(ev, data)	모든 연결된 클라이언트에 전송
socket.broadcast.emit()	발신자 제외 모든 클라이언트
io.to(room).emit()	룸의 모든 클라이언트
socket.to(room).emit()	발신자 제외 룸 멤버

브로드캐스트

브로드캐스트 메서드

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

취발성 및 압축

socket.volatile.emit()	클라이언트 준비 안 됐으면 버림 (버퍼링 없음)
socket.compress(true).emit()	메시지별 압축 활성화
io.local.emit()	로컬 서버에만 브로드캐스트 (다중 노드)
socket.timeout(5000).emit()	확인응답에 타임아웃을 설정하여 발신

룸

룸 작업

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

룸 속성

socket.rooms	이 소켓이 속한 룸 집합
socket.id	각 소켓은 자신의 ID 룸에 자동 참가
io.sockets.adapter.rooms	모든 룸과 멤버의 맵

룸 패턴

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

네임스페이스

네임스페이스 생성

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

클라이언트의 네임스페이스 연결

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

동적 네임스페이스

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

미들웨어

서버 미들웨어

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

네임스페이스 미들웨어

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

미들웨어 속성

socket.handshake.auth	클라이언트에서 전송된 인증 데이터
socket.handshake.headers	초기 요청의 HTTP 헤더
socket.handshake.query	연결 URL의 쿼리 파라미터
socket.data	미들웨어에서 첨부한 임의 데이터

에러 처리

서버 측 에러

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

클라이언트 측 에러

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

클라이언트 재연결 옵션

reconnection	자동 재연결 활성화 (기본 true)
reconnectionAttempts	최대 시도 횟수 (기본 무한)
reconnectionDelay	초기 지연 ms (기본 1000)
reconnectionDelayMax	최대 지연 ms (기본 5000)

Socket.IO 빠른 참조

확인응답

클라이언트 발신, 서버 확인

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

서버 발신, 클라이언트 확인

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

타임아웃 포함

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

일반 패턴

채팅 룸

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

온라인 상태

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

속도 제한

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