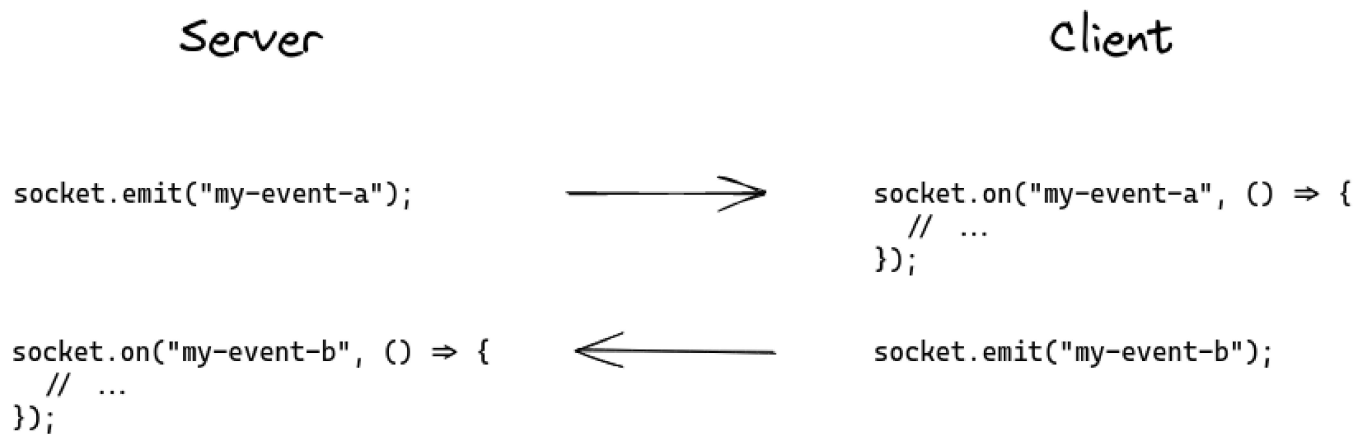
Version: 4.x

On this page

The Socket instance (server-side)

A `Socket` is the fundamental class for interacting with the client. It inherits all the methods of the Node.js EventEmitter, like emit, on, once or removeListener.


Server
Client
socket.emit("my-event-a");
socket.on("my-event-a", () => {
  // ...
});
socket.on("my-event-b", () => {
  // ...
});
socket.emit("my-event-b");

Besides:

- emitting and listening to events
- broadcasting events
- joining and leaving rooms

The Socket instance has a few attributes that may be of use in your application:

Socket#id

Each new connection is assigned a random 20-characters identifier.

This identifier is synced with the value on the client-side.

```
// server-side
io.on("connection", (socket) => {
  console.log(socket.id); // ojIckSD2jqNzOqIrAGzL
});

// client-side
socket.on("connect", () => {
  console.log(socket.id); // ojIckSD2jqNzOqIrAGzL
});
```

CAUTION

The `id` attribute is an **ephemeral** ID that is not meant to be used in your application (or only for debugging purposes) because:

- this ID is regenerated after each reconnection (for example when the WebSocket connection is severed, or when the user refreshes the page)
- two different browser tabs will have two different IDs
- there is no message queue stored for a given ID on the server (i.e. if the client is disconnected, the messages sent from the server to this ID are lost)

Please use a regular session ID instead (either sent in a cookie, or stored in the localStorage and sent in the `auth` payload).

See also:

- Part II of our private message guide
- How to deal with cookies

Note: you can't overwrite this identifier, as it is used in several parts of the Socket.IO codebase.

Socket#handshake

This object contains some details about the handshake that happens at the beginning of the Socket.IO session.

```
{
  headers: /* the headers of the initial request */
  query: /* the query params of the initial request */
  auth: /* the authentication payload */
  time: /* the date of creation (as string) */
  issued: /* the date of creation (unix timestamp) */
  url: /* the request URL string */
  address: /* the ip of the client */
  xdomain: /* whether the connection is cross-domain */
  secure: /* whether the connection is secure */
}
```

Example:

```
{
  "headers": {
    "user-agent": "xxxx",
    "accept": "*/*",
    "host": "example.com",
    "connection": "close"
  },
  "query": {
    "EIO": "4",
    "transport": "polling",
    "t": "NNjNltH"
  },
  "auth": {
    "token": "123"
  },
  "time": "Sun Nov 22 2020 01:33:46 GMT+0100 (Central European Standard
Time)",
  "issued": 1606005226969,
  "url": "/socket.io/?EIO=4&transport=polling&t=NNjNltH",
  "address": "::ffff:1.2.3.4",
  "xdomain": false,
  "secure": true
}
```

Socket#rooms

This is a reference to the rooms the Socket is currently in.

```
io.on("connection", (socket) => {
  console.log(socket.rooms); // Set { <socket.id> }
  socket.join("room1");
  console.log(socket.rooms); // Set { <socket.id>, "room1" }
});
```

Socket#data

An arbitrary object that can be used in conjunction with the `fetchSockets()` utility method:

```
// server A
io.on("connection", (socket) => {
  socket.data.username = "alice";
});

// server B
const sockets = await io.fetchSockets();
console.log(sockets[0].data.username); // "alice"
```

More information here.

Socket#conn

A reference to the underlying Engine.IO socket (see here).

```
io.on("connection", (socket) => {
  console.log("initial transport", socket.conn.transport.name); // prints
"polling"

  socket.conn.once("upgrade", () => {
    // called when the transport is upgraded (i.e. from HTTP long-polling to
WebSocket)
    console.log("upgraded transport", socket.conn.transport.name); // prints
"websocket"
  });

  socket.conn.on("packet", ({ type, data }) => {
    // called for each packet received
  });

  socket.conn.on("packetCreate", ({ type, data }) => {
    // called for each packet sent
  });

  socket.conn.on("drain", () => {
    // called when the write buffer is drained
  });
  socket.conn.on("close", (reason) => {
    // called when the underlying connection is closed
  });
});
```

Additional attributes

As long as you do not overwrite any existing attribute, you can attach any attribute to the Socket instance and use it later:

```
// in a middleware
io.use(async (socket, next) => {
  try {
    const user = await fetchUser(socket);
    socket.user = user;
  } catch (e) {
    next(new Error("unknown user"));
  }
});

io.on("connection", (socket) => {
  console.log(socket.user);

  // in a listener
  socket.on("set username", (username) => {
    socket.username = username;
  });
});
```

Socket middlewares

Those middlewares looks a lot like the usual middlewares, except that they are called for each incoming packet:

```
socket.use(([event, ...args], next) => {
  // do something with the packet (logging, authorization, rate limiting...)
  // do not forget to call next() at the end
  next();
});
```

The `next` method can also be called with an error object. In that case, the event will not reach the registered event handlers and an `error` event will be emitted instead:

```
io.on("connection", (socket) => {
  socket.use(([event, ...args], next) => {
    if (isUnauthorized(event)) {
      return next(new Error("unauthorized event"));
    }
    next();
  });

  socket.on("error", (err) => {
    if (err && err.message === "unauthorized event") {
      socket.disconnect();
    }
  });
});
```

Note: this feature only exists on the server-side. For the client-side, you might be interested in catch-all listeners.

Events

On the server-side, the Socket instance emits two special events:

- `disconnect`
- `disconnecting`

`disconnect`

This event is fired by the Socket instance upon disconnection.

```
io.on("connection", (socket) => {
  socket.on("disconnect", (reason) => {
    // ...
  });
});
```

Here is the list of possible reasons:

Reason	Description
`server namespace disconnect`	The socket was forcefully disconnected with socket.disconnect().
`client namespace disconnect`	The client has manually disconnected the socket using socket.disconnect().
`server shutting down`	The server is, well, shutting down.
`ping timeout`	The client did not send a PONG packet in the `pingTimeout` delay.
`transport close`	The connection was closed (example: the user has lost connection, or the network was changed from WiFi to 4G).
`transport error`	The connection has encountered an error.
`parse error`	The server has received an invalid packet from the client.
`forced close`	The server has received an invalid packet from the client.
`forced server close`	The client did not join a namespace in time (see the `connectTimeout` option) and was forcefully closed.

`disconnecting`

This event is similar to `disconnect` but is fired a bit earlier, when the Socket#rooms set is not empty yet

```
io.on("connection", (socket) => {
  socket.on("disconnecting", (reason) => {
    for (const room of socket.rooms) {
      if (room !== socket.id) {
        socket.to(room).emit("user has left", socket.id);
      }
    }
  });
});
```

Note: those events, along with `connect`, `connect_error`, `newListener` and `removeListener`, are special events that shouldn't be used in your application:

```
// BAD, will throw an error
socket.emit("disconnect");
```

Complete API

The complete API exposed by the Socket instance can be found here.

Edit this page

*Last updated on **12/19/2022***