Broadcast to Multiple Clients

11 min

Nice work! Now that you've practiced sending and responding to messages from the client and the server, you're ready to make your chat application a little more functional. Right now, when a client messages the server, the server echoes the message back, but only to the sender. This setup makes for a somewhat lonely chat room 😢.

Ideally, when one client messages the server, all other clients should hear about it. To save your users from chatting with themselves in an echo chamber forever, you'll have to implement *broadcasting*. Broadcasting is a common pattern in WebSocket applications in which a server sends the same message to multiple connected clients.

Broadcasting works like this: upon receiving a message from a client, a WebSocket server:

- Iterates through the complete list of its connected clients, and for each connected client:
    - Checks if the client's connection is still open:
        - If so, sends the broadcast message to that client
    - Continues to the next client

Let's consider how we might implement this pattern by attempting to announce to all members of the chat room when a new user joins.

First, a server must iterate through all of its connected clients. Conveniently, a WebSocketServer has a .clients property that holds an array of the WebSocket connection objects for each client connected to the server. Iterating through this client list might look like this:

```
// Server
wsServer.on('connection', socket => {
  // iterate over all clients connected to wsServer
  wsServer.clients.forEach(connectedClient => {
    // send the broadcast message
  })
})
```

As we iterate through each connectedClient in the .clients array, we want to make sure that the client has an open connection before sending the message to that client. To make this check, each connection object has a .readyState property that indicates whether the client is connecting (0), is actively open (1), is closing (2) or is closed (3).

To facilitate the writing of clean code (eg. so as to encourage people not to hard-code the numbers corresponding to these ready states in their applications) the default WebSocket object exported by ws has a set of constant values corresponding to the four ready states that a WebSocket client may be in. They are:

- WebSocket.CONNECTING whose value is 0
- WebSocket.OPEN whose value is 1
- WebSocket.CLOSING whose value is 2
- WebSocket.CLOSED whose value is 3

Putting this together might look like this:

```
// Server
wsServer.on('connection', socket => {
  // iterate over all clients connected to wsServer
  wsServer.clients.forEach(connectedClient => {
    // check if the connection is open
    if (connectedClient.readyState === WebSocket.OPEN) {
      // if so, send the broadcast message
      connectedClient.send('Hear ye hear ye. A new user has joined!'');
    }
  })
})
```

Since each connectedClient is a WebSocket connection object it has a .send() method which is used to ultimately send a message directly to the specified client.

Often, the socket emitting the message will be left out of the broadcast. Before broadcasting a message, the server can check to see if the current client is the same as the socket that emitted the message by directly comparing the current connectedClient with the socket that opened the connection:

```
// Server
wsServer.on('connection', (socket) => {
  // iterate over all clients connected to wsServer
  wsServer.clients.forEach(connectedClient => {
    // check if the connection is open AND is not the emitting socket.
    if (connectedClient.readyState === WebSocket.OPEN && connectedClient !== socket) {
      // if so, send the broadcast message
      connectedClient.send('Hear ye hear ye. A new user has joined!'');
    }
  });
});
```

This is certainly not the only way to implement this pattern, however it demonstrates a few key features of the ws package:

- Instances of the WebSocketServer class (wsServer in this case), have a .clients array holding all of the WebSocket connection

Preview: Docs Loading link description

[objects]

.

- Each WebSocket connection object has a .readyState property that indicates whether the connection is connecting, open, closing, or closed.
- The default export of the ws package has a set of constant values corresponding to the possible ready states of a WebSocket connection object.

Instructions

Task 1

Broadcasting is such a common pattern in WebSockets that it will be useful to write a function for it. That way, we can easily reuse the logic shown above throughout our program. Near the bottom of server.js, an outline for such a broadcast() function has been created for you. It accepts two arguments—the data to broadcast, and a socket to omit from the broadcast (typically, the socket that sent the message).

Implements the broadcast pattern inside this function. Make sure to exclude the emitting socket from the message being broadcast.

Help

You can use !== to compare each socket against the emitting socket.

Task 2

Back inside the 'message' event handler defined for the connected socket, call broadcast() so that, rather than echoing a client's message back to just the emitting socket, your server sends the new message to all other connected clients.

Help

Remember, broadcast() accepts two arguments – the data to be broadcast as well as the emitting socket, which should be omitted from the broadcast.

Task 3

Restart your server and then test that your code works by opening two browser windows and sending a message from one. At this point, your message should appear in the other!

Help

Press Control+c to stop the server and then enter the node server.js command again.