

CSCI 4140 – Tutorial 8

WebSocket and Socket.IO

Matt YIU, Man Tung ([mtyiucse](mailto:mtyiucse@gmail.com))

SHB 118

Office Hour: Tuesday, 3-5 pm

2015.03.12

Outline

- What is WebSocket?
- What is Socket.IO?
- Get started with a chat application
- Namespaces and rooms in Socket.IO
- Socket.IO in Assignment 2

What is WebSocket?

- A protocol providing **full-duplex** (read & write) communications channels over a single **TCP connection**
- Designed to be implemented in **web browsers** and **web servers**
- A **dedicated server** is needed because an **application-level handshaking** is needed
- Other than that, WebSocket programming is the same as ordinary **socket programming**
- URI scheme: **ws:** and **wss:** for unencrypted and encrypted connections respectively (just like http: and https:)

What is Socket.IO?

- A JavaScript library for **realtime** web applications
- It enables **real-time bidirectional event-based** communications
- It primarily uses the **WebSocket** protocol with **polling** as a **fallback option**
 - It provides many more features than WebSocket, e.g., **broadcasting** to multiple sockets, storing data **associated with each client**, and **asynchronous I/O**
- It has two parts:
 - A **client-side** library that runs in the browser
 - A **server-side** library for Node.js
- Can be installed with the **npm** tool

Adapted from <http://socket.io/get-started/chat/>

Get started with a chat application

Learning the basics of Socket.IO through an chat application!

Create an Express application skeleton

- Let's use the **Express** framework for simplicity
- Create an Express application called “socket-io-chat” and install dependencies:

```
$ express socket-io-chat  
(Output omitted)  
$ cd socket-io-chat  
$ npm install  
(Output omitted)
```

“Hello World” with `index.js`

- Setup our application by creating `index.js`:

```
var app = require( 'express' )();

app.get( '/', function( request, response ) {
  response.send( 'Hello World' );
} );

var server = app.listen( 4140, function() {
  var host = server.address().address;
  var port = server.address().port;

  console.log( 'Listening on http://%s:%s...', host, port );
} );
```

`index.js`

- Run “`node index.js`” and visit <http://127.0.0.1:4140/>

Implement the UI

- Implement the chat room user interface in HTML (**views/index.html**)
 - Download the HTML from the example code (**views/index-begin.html**)
- Serving HTML in Express:

```
app.get( '/', function( request, response ) {  
    response.sendFile( __dirname + '/views/index.html' );  
} );
```

Change this line in **index.js**

index.js

- Restart the Node process and refresh the page

Integrating Socket.IO

- Socket.IO is composed of two parts:
 - A server that integrates with (or mounts on) the **Node.js HTTP Server: `socket.io`**
 - A client library that loads on the browser side: **`socket.io-client`**
 - This library is served to the client **automatically**
- Before using the library, we need to install it using npm

```
$ npm install --save socket.io
```

 - That will **install the module** and **add the dependency** to **`package.json`**

Integrating Socket.IO

- Integrate Socket.IO into **index.js**

```
// ... (omitted)
var server = app.listen( 4140, function() {
  var host = server.address().address;
  var port = server.address().port;

  console.log( 'Listening on http://%s:%s...', host, port );
} );

var io = require( 'socket.io' )( server );
io.on( 'connection', function( socket ) {
  console.log( 'New user connected' );
} );
```

Add these lines to
the end of the file.

index.js

Integrating Socket.IO

- What do these lines do?

Initialize a **socket.io** instance by passing the **server** object.

```
var io = require( 'socket.io' )( server );  
io.on( 'connection', function( socket ) {  
  console.log( 'New user connected' );  
} );
```

Listen on the **connection** event for incoming sockets.

The signature of the event listener is:

```
function (socket) { /* ... */ }
```

Integrating Socket.IO

- Integrate Socket.IO into **views/index.html**

```
<!-- ... (omitted) ... →  
    <script src="/socket.io/socket.io.js"></script>  
    <script>  
        var socket = io();  
    </script>  
</body>  
</html>
```

Add these lines before **</body>**.

views/index.html

- The first line loads the **socket.io-client** library which exposes an **io** global
 - Call **io()** without specifying any URL means to connect to the host that serves the page
- Now reload the server and refresh the web page

Integrating Socket.IO

- Try opening several tabs
- Can you see the message “**New user connected**” in the terminal?
- Each socket also fires a special **disconnect** event:

```
// ... (omitted)
var io = require( 'socket.io' )( server );
io.on( 'connection', function( socket ) {
  console.log( 'New user connected' );
  socket.on( 'disconnect', function() {
    console.log( 'User disconnected' );
  } );
} );
```

index.js

Add these lines into **index.js** and reload the server. You can see “**User disconnected**” upon each disconnection.

Emitting a chat event

- You can send (or emit) and receive any events, with any data in Socket.IO
- Let's emit an “**chat**” event when the user types in a message
- Modify the last **<script>** tag in **views/index.html**:

```
<script>
  var socket = io();
  var form = document.querySelector( '#form' );
  var m = document.querySelector( '#m' );
  form.addEventListener( 'submit', function( e ) {
    e.preventDefault();
    socket.emit( 'chat', m.value );
    m.value = '';
  } );
</script>
```

views/index.html

Emitting a chat event

- You can send (or emit) and receive any events, with any data in Socket.IO
- Let's emit an “**chat**” event when the user types in a message
- Modify the last **<script>** tag in **views/index.html**:

```
<script>
  var socket = io();
  var form = document.querySelector( '#form' );
  var m = document.querySelector( '#m' );
  form.addEventListener( 'submit', function( e ) {
    e.preventDefault();
    socket.emit( 'chat', m.value );
    m.value = '';
  } );
</script>
```

Get the DOM element using `querySelector()`.

Add an event listener for the form's **submit** event.

Emit a “**chat**” event with the message (`m.value`) as the data with Socket.IO

x.html

Emitting a chat event

- Use `socket.on(<event>, function(data) { /* ... */ })` to handle our newly defined event

```
// ... (omitted)
var io = require( 'socket.io' )( server );
io.on( 'connection', function( socket ) {
  console.log( 'New user connected' );
  socket.on( 'disconnect', function() {
    console.log( 'User disconnected' );
  } );
  socket.on( 'chat', function( data ) {
    console.log( 'Message: ' + data );
  } );
} );
```

`index.js`

Add these lines into `index.js` and reload the server. You can see the message from the client upon each form submit.

Broadcasting

- Next, we need to emit the event from the server to **all connected users** such that they can see the message
- Modify the **chat** event listener:

```
// ... (omitted)
socket.on( 'chat', function( data ) {
  console.log( 'Message: ' + data );
  io.emit( 'chat', data );
} );
```

index.js

A **chat** event is emitted to all connected clients with the **data**.

Broadcasting

- Listen to the **chat** event in the client side:

```
// ... (omitted)
var messages = document.querySelector( '#messages' );
socket.on( 'chat', function( data ) {
    var li = document.createElement( 'li' );
    li.innerHTML = data;
    messages.appendChild( li );
} );
```

views/index.html

Broadcasting

- Listen to the **chat** event in the client side:

Set up a **chat** event listener to the **socket**.

The signature of the event listener is:

```
function ( data ) { /* ... */ }
```

```
// ... (omitted)  
var messages = document.querySelector( '#messages' );  
→ socket.on( 'chat', function( data ) { ←  
    var li = document.createElement( 'li' );  
    li.innerHTML = data;  
    messages.appendChild( li );  
} );
```

Display the incoming message by updating the DOM tree.

views/index.html

- That completes our chat application!
 - It supports multiple clients!



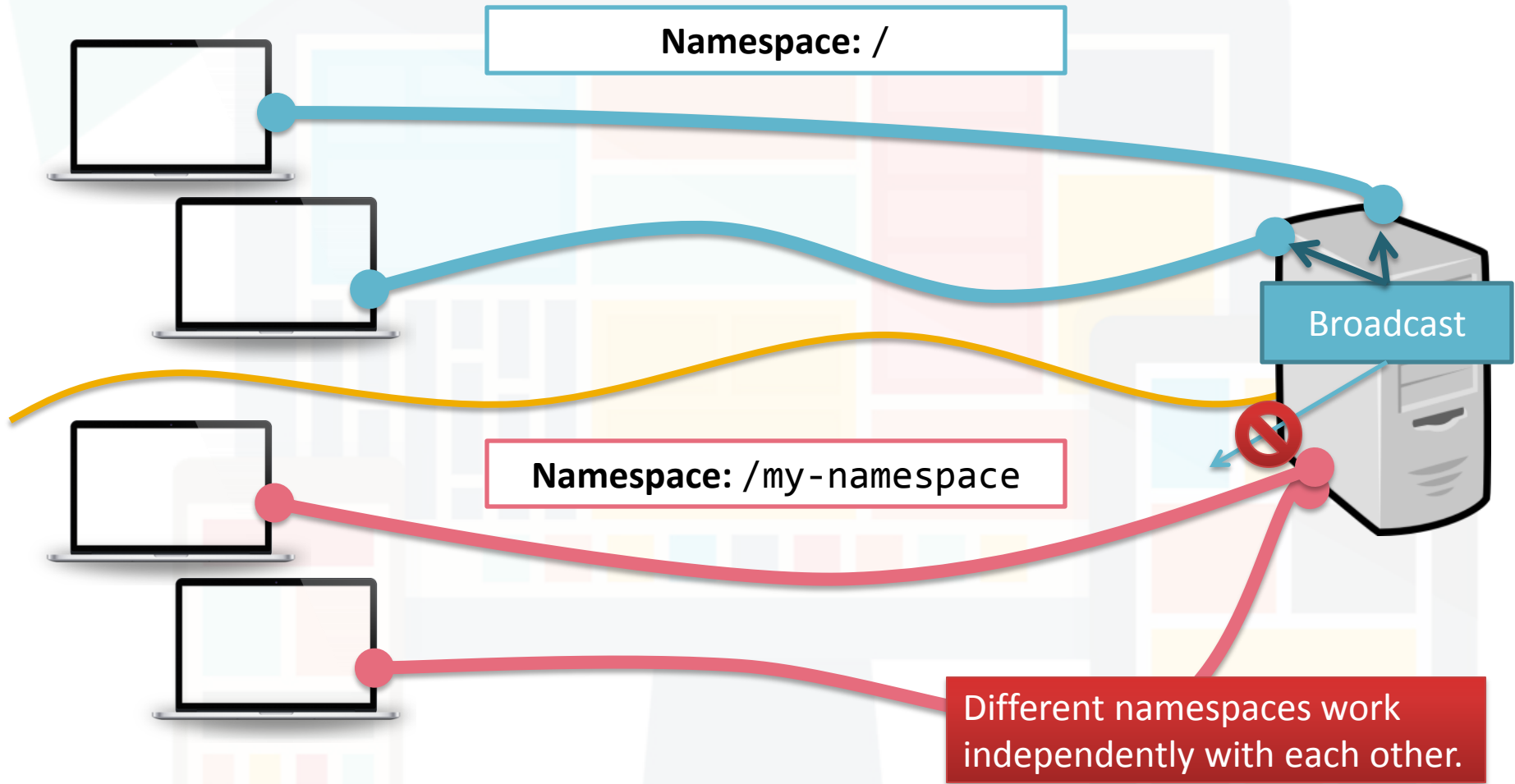
Namespaces and rooms in Socket.IO

We can broadcast among clients in the same namespace / room only!

Namespaces

- Socket.IO allows you to “**namespace**” your sockets, which essentially means assigning different **endpoints** or **paths**
- Useful for
 - **Minimizing** the number of **resources** (e.g., TCP connections)
 - Introducing **separation** between **communication channels**
- The default namespace is “/”
 - The clients connect to this namespace by default
 - The server listens to this namespace by default

Namespaces



Custom namespaces

- To set up a custom namespace, call the **of** function on the **server-side**:

```
var nsp = io.of( '/my-namespace' );  
nsp.on( 'connection', function ( socket ) {  
  console.log( 'someone connected' );  
});  
nsp.emit( 'hi', 'everyone!' );
```

- On the **client side**, specify the namespace in the **io** function:

```
var socket = io( '/my-namespace' );
```

For your information, my implementation does not use custom namespaces to separate different sessions. I use “room” instead!

Rooms

- Within each namespace, you can also define arbitrary channels (denoted as “**room**”) that sockets can **join** and **leave**
- To assign the sockets into different rooms on the server side:

```
io.on( 'connection', function( socket ) {  
  socket.join( 'some room' );  
} );
```

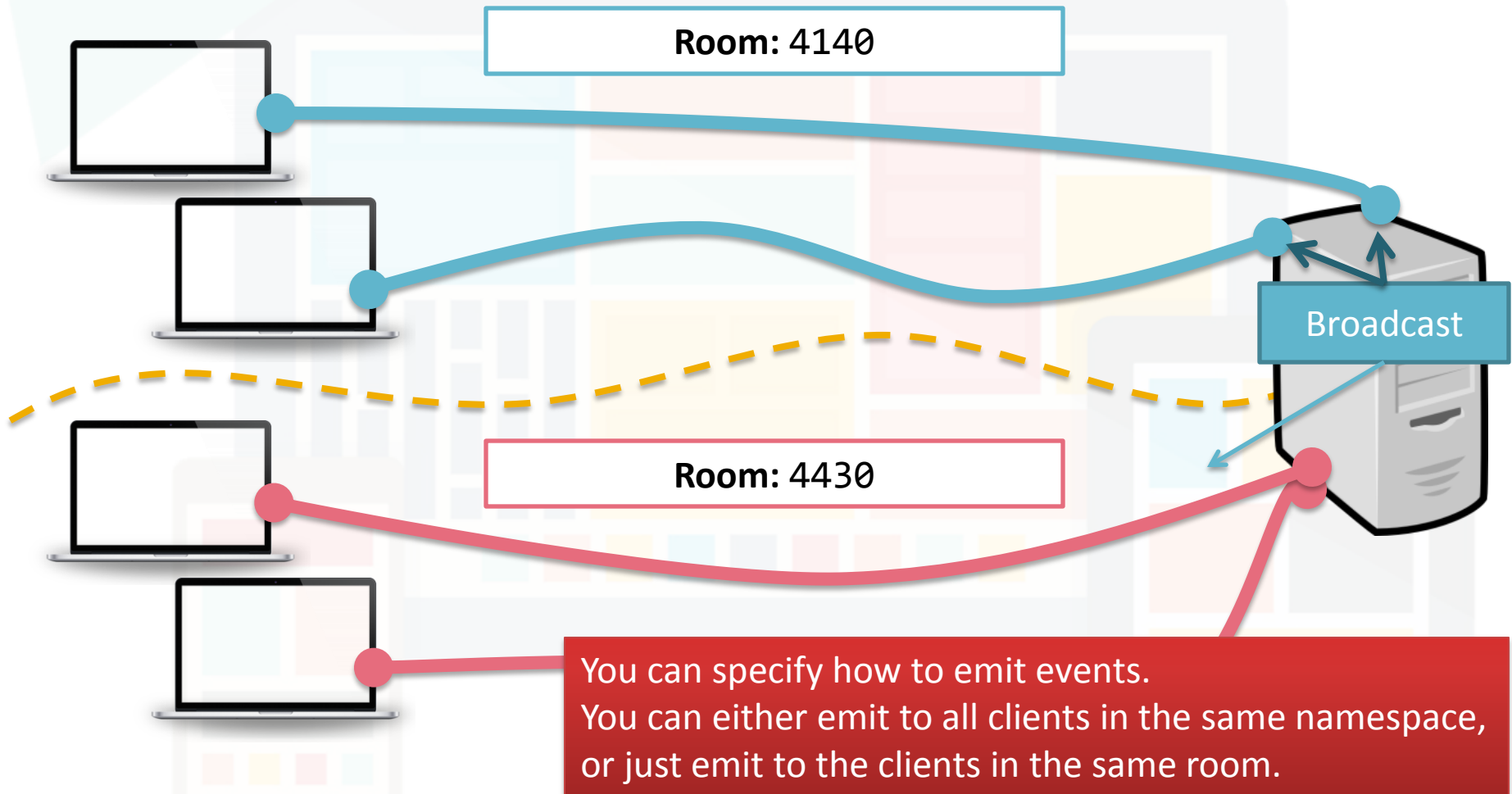
Of course, you can join (i.e., subscribe the socket to a given channel) when other events are emitted, e.g., “**register**” event

- To broadcast or emit using **to** and **in**:

```
io.to( 'some room' ).emit( 'some event' );
```

- To leave a channel: **socket.leave**('some room');
 - This is automatically done upon disconnection

Rooms under the same namespace





Socket.IO in Assignment 2

Socket.IO is the core of the remote control!

Socket.IO in Assignment 2

- Socket.IO is used for
 - **Connecting** the clients to the server
 - **Broadcasting control signals** to the desktop clients
 - **Synchronizing** the playlist
- Emitted events in my implementation (*for your reference only*)
 - **register** (*data*: session ID) – Assign a socket to a room
 - **sync / download / upload** (*data*: null or playlist) – Playlist synchronization request and response
 - **command** (*data*: control signal to the player)
 - **add / remove** (*data*: video ID to be added or removed)
 - **Feel free to design your own protocol!**

References

- Get Started: Chat application
 - <http://socket.io/get-started/chat/>
- Server API
 - <http://socket.io/docs/server-api/>
- Client API
 - <http://socket.io/docs/client-api/>
- Rooms and Namespaces
 - <http://socket.io/docs/rooms-and-namespaces/>

– End –