Real-time Communication using Node.js & Socket.IO

Presenter: Vageesh Bhasin, Mindfire Solutions
Date: 20 – June – 2014

Agenda

- WebSocket
  - Introduction to WebSockets
  - How WebSockets work
- Socket.IO
  - Introduction to Socket.IO
  - Using Socket.IO
  - Firing events
  - Listening to events
- Demo on creating a small app

WebSockets

Introduction to WebSocket

- Protocol providing **full-duplex communication** channel over single **TCP connection**
- Web was built around the idea – **'Client requests, Server fulfills'**
- AJAX got people started to look for ***bidirectional*** connections
- Other strategies like **long-polling** had the problem of **carry overhead**, leading to increase in latency

How does WebSockets Work?

- Establish connection through 'WebSocket Handshake'
  - Client Request

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Origin: http://example.com
```

  - Server Response

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
```

- After handshake, initial HTTP connection is replaced by WebSocket connection (uses same underlying TCP/IP)
- Transfer data without incurring any overhead associated with requests

Socket.IO

Introduction to Socket.IO

- **JavaScript library** for **realtime** web applications
- Has two parts:
  - **Client-side** – Runs on Browser
  - **Server-side** – Runs on Server – Node.js
- Primarily uses **WebSocket**, but can **fallback** to other methods like **AJAX long** polling, **JSONP** polling
- In addition to one-to-one communication as in WebSocket, it enables **broadcasting** to multiple nodes

Using Socket.IO

- **With Node HTTP Server**

```
// app.js
// Requiring Module Dependencies

var app = require('http').createServer(serveFile),
io = require('socket.io')(app),
fs = require('fs');

app.listen(3000, function () {
  console.log('Server up and listening to port 3000');
});

// Handler to serve static files
function serveFile(req, res) {
        fs.readFile(__dirname + '/index.html', function(err, data) {
                if(err) {
                        res.writeHead(500);
                        return res.end('Error loading index.html');
                }
                res.writeHead(200);
                return res.end(data);
        });
}

// Socket.IO
io.on('connection', function (socket) {
        // Event 1
        socket.emit('event_1', {hello: 'world!'});
        // Event 2
        socket.on('event_2', function(data) {
                console.log(data);
        });
});
```

```
// index.html

<html>
<head>
        <title>Socket.io Demo</title>
        <script src="/socket.io/socket.io.js"></script>
        <script>
                var socket = io('http://localhost');
                socket.on('event_1', function (data) {
                        console.log(data);
                        socket.emit('event_2', { my: 'data' });
                });
        </script>
</head>
<body>
        <h1>Socket.IO Demo</h1>
</body>
</html>
```

Using Socket.IO

- **With Express.JS**

```
// Requiring Module Dependencies

var app = require('express')(),
server = app.listen(3000, function () {
  console.log('Server up and listening to port 3000');
}),
io = require('socket.io').listen(server);

app.get('/', function(req, res) {
        res.sendfile(__dirname + '/index.html');
});

// Socket.IO
io.on('connection', function (socket) {
        // Event 1
        socket.emit('event_1', {hello: 'world!'});
        // Event 2
        socket.on('event_2', function(data) {
                console.log(data);
        });
});
```

```
// index.html

<html>
<head>
        <title>Socket.io Demo</title>
        <script src="/socket.io/socket.io.js"></script>
        <script>
                var socket = io('http://localhost');
                socket.on('event_1', function (data) {
                        console.log(data);
                        socket.emit('event_2', { my: 'data' });
                });
        </script>
</head>
<body>
        <h1>Socket.IO Demo</h1>
</body>
</html>
```

Firing Events

- Individual Recipient - **EMIT**
  - Current connected socket

    SYNTAX: socket.emit('eventName', "Event Data");

  - Specific Socket

    SYNTAX: io.sockets.socket(socketId).emit('eventName', "Event Data");

- Multiple Recipients – **BROADCAST**
  - All connected nodes **except** the current one

    SYNTAX: socket.broadcast.emit('eventName', "Event Data");

  - All connected nodes

    SYNTAX: io.sockets.emit('eventName', "Event Data");

- Specific Channel – **TO/IN**
  - To all connected nodes in a channel **except** current

    SYNTAX: socket.broadcast.to(channelName).emit('eventName', "Event Data");

  - To all connected nodes in a channel

    SYNTAX: io.sockets.in(channelName).emit('eventName', "Event Data");

Listening to Events

- Listening to events is easier as compared to firing events

Syntax:

```
socket.on('eventName', handler);
```

Example:

```
// Register a handler to listen to 'event_1'
socket.on('event_1', function (data) {
        // Respond by sending message with time stamp to all nodes
        io.sockets.emit('pushMessage', {
                message: data.message,
                time: new Date()
        });
});
```

Thank you