CSC309 Winter 2016
Lecture 9

Larry Zhang

Announcement

- A3 proposal due this evening.
- Don’t forget to give your game a catchy name
- We have iPad mini’s, iPod touch’s and Nexus phones and tablets which you can borrow for the project, ask me for permission first, then get it from Andrew Wang.


```
Part of WebSocket Interface:
attribute Function onopen;
attribute Function onmessage;
attribute Function onerror;
attribute Function onclose;
```

HTML5 WebSocket

- Standardized by IETF in 2011.
- Supported by most major browsers including Google Chrome, Internet Explorer, Firefox, Safari and Opera
- Basically, it allows the browser to create a **socket client** (**inside a web page**) that connects and talks to a WebSocket server.

Why WebSocket

- **Low latency client-sever and server-client connections.**

- **Legacy HTTP**
  - Connect, send a request, get a response, disconnect. When sending the next request need to establish the connection again.
  - All communications are initiated by the client, the server cannot proactively send something to the client.
- **WebSocket**
  - full-duplex communication channel over a single always-on TCP connection

Legacy HTTP vs WebSocket

http://

ws://

Choose the right thing

- HTTP is still great for static, cachable content
- WebSocket is better for real-time applications, like games, stock monitors

How the protocol works

- It starts off as a HTTP request, which indicates that it wants to “upgrade” to the WebSocket protocol

- If you server can understand it, then the http connection is switched into a WebSocket connection

request

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

response

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

How to use WebSocket

- server side
- client side

Server side

- Use ***ws***, the WebSocket server implementation for Node.js
  - https://github.com/websockets/ws
- Other server implementation also exists, for different languages, e.g., Java, Python, etc.

Example: simple echo server

```
 1
 2 var WebSocketServer = require('ws').Server
 3     ,wss = new WebSocketServer({port: 10000});
 4
 5 wss.on('close', function() {
 6     console.log('disconnected');
 7 });
 8
 9 wss.on('connection', function(ws) {
10     ws.on('message', function(message) {
11         console.log(message);
12         ws.send(message);
13     });
14 });
15
```

ws is new socket for the new connection

Client side: basic structure

```
socket = new WebSocket("ws://localhost:8000");

socket.onopen = function (event) {}

socket.onclose = function (event) {
    event.code
    event.reason
    event.wasClean
}

socket.onmessage = function (event) {
    event.data
}
```

Everything is event-driven

Notes

- Client and server communicate by sending messages to each other.
- Server can easily broadcast message to all its clients.
- If you run a web page “client.html” on CSLINUX.UTM, make sure you’re visiting “**http**://cslinux…/client.html”, rather than “**https**://cslinux…/clien.html”.
  - you’re supposed to use secure ws, i.e., wss, with https, but that doesn’t work on cslinux.

demo

http://www.cs.toronto.edu/~ylzhang/csc309w16/websocket/

Reference

- https://www.websocket.org/
- https://github.com/websockets/ws
- https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API