WebSocket Tutorial with Node.js

by Joshua Kehn on 2010-09-22 1:48 pm

> ***Note this is a re-write of an earlier post that got lost due to data corruption.***

WebSockets ***were*** part of the HTML5 **draft specifications** but have since been moved into their own section. Currently supported by Google **Chrome**, Apple **Safari**, and **Firefox 4.0** (beta). Support for IE is not forthcoming, which leaves about 70% of the browsers in the dark. WebSockets provide some amazing opportunities for development, especially when working with live / streaming data. What do we do?

Introduce the **Flash Socket**. Flash Sockets are native socket connections. They have a few limitations (must have flash, flash policy file) but they are well worth it to ensure that the majority of people have access to your application. Currently this is supported in IE 8+, FF 3+, and Chrome, as well as any browser that supports native WebSockets. I've also had good success with Safari, but the goal here was IE support, which we now have.

What you will need

Starting from the client side and progressing to the server:

- Node.js (more below)
- **web-socket-js** (client side Flash Sockets)
- **node-websocket-server** (Node WebSocket implementation)
- Flash policy file server (I will explain how to make this)
- Something to server the HTML page, as you can't use a `file:///` URI with Flash Sockets.

Node.js

Node.js is a evented I/O framework built around Google's **V8 JavaScript Engine**. You can download it from the **ry / node git repository**. Installing is pretty simple on most Linux / Unix systems. I've tested this under Ubuntu 10.04 and Mac OS X 10.6.x;

```
git clone http://github.com/ry/node
cd node/
./configure
sudo make install
```

Expect the configure to go quickly, and the install to take upwards of six to eight minutes. After this you would be able to run `node --version` and receive the version string. If not, I refer you to Node's **Google Group** for help installing Node. **Do not proceed until you have Node installed.**

Build

We will be building a very basic application that sends messages spaced one second apart containing the current date to the web page. While the practical applications for this are limited, it is a simple, yet effective, introduction and proof-of-concept. The concept here being that WebSockets are available cross browser and easy to develop.

Client

The client side is particularly easy. There are three files you need to include, `swfobject.js`, `FABridge.js`, and `web_socket.js`. Then you need to write a short `script` section that will set the swf location.

```
<html>
<head>
<title>WebSocket demo</title>
<!-- JS files for Flash Socket -->
<script type="text/javascript" src="js/swfobject.js"></script>
<script type="text/javascript" src="js/FABridge.js"></script>
<script type="text/javascript" src="js/web_socket.js"></script>
<script type="text/javascript">
/**
* Set the swf file location
*/
WEB_SOCKET_SWF_LOCATION = "swf/WebSocketMain.swf";
</script>
```

After that is done we can setup our WebSocket code and handlers.

```
var ws = new WebSocket("ws://localhost:40132");
ws.onopen = function()
{
output("Opened connection.");
};
ws.onmessage = function(e)
{
date = new Date(e.data);
if(isNaN(date.getHours()))
{
output(e.data);
}
else
{
var hours = date.getHours();
var minutes = date.getMinutes();
var seconds = date.getSeconds();
var ap = "AM";
if(hours > 12)
{
hours = hours - 12;
ap = "PM";
}
if(hours < 10)
{
hours = "0" + hours;
}
if(minutes < 10)
{
minutes = "0" + minutes;
}
if(seconds < 10)
{
seconds = "0" + seconds;
}
output("Currently " + hours + ":" + minutes + ":" + seconds + " " + ap);
}
};
ws.onclose = function()
{
output("Closed connection.");
};
ws.onerror = function()
{
output("Error occurred.");
};
```

Define the output function.

```
function output(str)
{
document.getElementById("log").innerHTML = str;
}
```

And the rest of the HTML page.

```
</script>
</head>
<body>
<div id="log"></div>
</body>
</html>
```

Server

Now we move on to the server side of things. I use Node.js and a really good **websocket implementation**, however there are implementations for **Ruby**, **Python**, and even **PHP**.

Flash Policy

Remember that **Flash Policy File** I mentioned earlier? Yep, have to implement it. Flash attempts to ensure security by checking two places, port 843 and the destination port. It's always good to run a separate Flash Policy Server because Flash player will ***by default*** attempt for three seconds on port 843 before moving on. This **will** delay your WebSocket by three seconds.

The Flash Policy Server (also node) is pretty basic.

```
var net = require("net"),
domains = ["*:*"];
```

`domains` is a paired array of domains and ports. The above will allow any connecting domain to connect to any port as a socket. If you desire more security, you can limit to your own domain and a specific port.

```
net.createServer(
function(socket)
{
/**
* Start the flash policy file
*/
socket.write("<?xml version=\"1.0\"?>\n");
socket.write("<!DOCTYPE cross-domain-policy SYSTEM \"http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd\">\n");
socket.write("<cross-domain-policy>\n");
/**
* Iterating over the domains array
*/
domains.forEach(
function(domain)
{
/**
* Break the domain into domain and port values
*/
var parts = domain.split(':');
/**
* Write them to the socket
*/
socket.write("<allow-access-from domain=\""+parts[0]+"\"to-ports=\""+(parts[1]||'80')+"\"/>\n");
}
);
/**
* And we're done
*/
socket.write("</cross-domain-policy>\n");
require("sys").log("Wrote policy file.");
socket.end();
}
).listen(843);
```

The Flash Policy server binds to a low port (843) so you will have run it under **sudo** or root.

```
sudo node flashpolicy.js > flashpolicy.log & sleep 5; tail flashpolicy.log
```

Ensure that you have a "Flash server listening" log line before continuing. After you have that setup you can work on the actual server side code.

Node.js

Server side isn't much more difficult, in fact it could be easier to understand then the client.

```
var sys = require("sys")
, ws = require('./lib/ws');
var server = ws.createServer();
intervals = {};
server.addListener("listening",
function()
{
sys.log("Listening for connections.");
}
);
```

Then how we handle WebSocket connections

```
// Handle WebSocket Requests
server.addListener("connection",
function(conn)
{
conn.send("Connection: "+conn.id);
sys.log("Connection found: " + conn.id);
```

We are going to add a message listener even though we aren't using that here.

```
conn.addListener("message",
function(message)
{
sys.log("Message recieved from conn# " + conn.id + "\n" + message);
}
);
```

Setup an interval (1000 msec = 1 second) to send the date.

```
intervals[conn] = setInterval(
function()
{
date = new Date();
conn.send(date.toString());
}
, 1000);
}
);
```

Closing clears the interval and broadcasts a message saying so.

```
server.addListener("close",
function(conn)
{
clearInterval(intervals[conn]);
server.broadcast("<"+conn.id+"> disconnected");
sys.log(conn.id + " disconnected");
}
);
```

Finally, listen to port 40132. Port number was pulled out of a hat.

```
server.listen(40132);
```

That wraps this up, if anything is confusing drop me a comment and I'll see what I can do to clear it up.

I have this "live" under **joshuakehn.com/socketexample** if you want to see it. You can also access this "shell style" with the following handshake:

```
GET /echo HTTP/1.1
Host: localhost
Connection: Upgrade
Upgrade: WebSocket
```

This will be answered with

```
HTTP/1.1 101 Web Socket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
WebSocket-Origin: undefined
WebSocket-Location: ws://joshuakehn.com/echo
WebSocket-Protocol: *
```

And follow with a list of times. Terminal output isn't pretty, but you get the gist of it.
