

Let's chat about

nodeJS

with express and socket.io

What is node anyway?

node is...

A way to program web apps in Javascript

A protocol-agnostic server platform

An asynchronous event-loop

Based on Chrome's V8 engine

Capable of high concurrency

A landscape photograph featuring a double rainbow arching across a cloudy sky. The scene is set in a rural area with a dirt path curving through a field. A single, leafless tree stands prominently in the middle ground. The lighting is soft, suggesting dawn or dusk, with the rainbow's colors appearing more muted and golden. The text "What does it mean?" is overlaid in the center in a white, serif font.

What does it mean?

It means...

You can make web apps that can easily handle thousands of connections at once, and you can program it in a fraction of the time, with less code and less horrid work-arounds.

Linearity is inefficient.

```
$path = 'file.txt';  
$mime = mime_content_type($path);  
$file = file_get_contents($path);  
header('Content-Type: '.$mime);  
echo $file;
```

While the contents of 'file.txt' are being collected, nothing else is being processed. Returning the content type header on the next line is not dependent on the file data.

Events are better.

```
var path = 'file.txt';
fs.read(path, function(err, data){
  res.write(data);
});
fs.stat(path, function(err, stat){
  res.writeHead(200, {
    'Content-Length': stat.size
  });
});
```

The callbacks for `fs.read()` and `fs.stat()` both execute at the same time. Processing cycles are never wasted.

node is object-oriented.

Node stores reusable functionality in modules.

Node comes with several modules including:

- http for creating web servers.

- tcp for creating tcp servers.

- fs for interacting with the file system.

To create a web server, we load the http module, then we supply a callback to `createServer()`, which will be called for every http request.

Let's make a basic "Hello World" test server.

```
// Load an HTTP server module.
var http = require('http');

// Create HTTP server.
http.createServer(function(req, res){
  // Write response chunk.
  res.write('Hello World\n');

  // Close connection.
  res.end();
});

// Listen for connections on port 80.
http.listen(80, "127.0.0.1");
```

Easy, right?
Lets make it easier.

Express is a handy third-party module which supplies common http functionality like URL routing, static file serving, and html templating.

It also handles lots of the little details for you.

Lets redo our "Hello World" with express.

```
// Generate an express server.
var express = require('express');
var app = express.createServer();

// Set route for '/'
app.get('/', function(req, res){
    res.send('Hello World');
});

// Listen on port 80.
app.listen(80);
```

Let's try a chatroom.

First of all, lets rewrite our express server.

This time, we don't even need a url route.

We just use `app.configure()` to serve static files.

```
// Generate an express server.
var express = require('express');
var app = express.createServer();

// Configure app to serve static files.
app.configure(function(){
  var path = __dirname + '/static';
  app.use(express.staticProvider(path));
});

// Listen on port 80.
app.listen(80);
```

The static file serving is out of the way,
now let's try setting up the socket.io
server for real-time communication.

First we need to load the `socket.io` module, and connect it to the existing http server.

```
// Create Socket.IO server.  
var socket = require('socket.io');  
  
// Attach it to the express server.  
socket.listen(app);
```

```
// Define connection event.
socket.on('connection', function(client){
  // Notify others when a user connects.
  client.broadcast('Someone logged in!');

  // Pass messages to other users.
  client.on('message', function(msg){
    client.broadcast(msg);
  });

  // Notify others when a user disconnects.
  client.on('disconnect', function(){
    client.broadcast('Someone logged out!');
  });
});
```

Next, let's create an `index.html` file and store it in the static folder defined in `app.configure()`.

First of all, we need the node-hosted socket.io script. Include this line somewhere in the body of index.html.

```
<script src="/socket.io/socket.io.js"></script>
```

Now we need to initialize the socket.io connection and define event handlers. Don't worry, it's super easy.

```
<script src="/socket.io/socket.io.js"></script>
<script>
// Create Socket.IO instance.
var socket = new io.Socket('localhost');

// Find log element to append messages to.
var log = document.getElementById('log');

// Append received messages to log.
socket.on('message', function(msg){
    log.innerHTML += '<p>'+msg+'</p>';
});

// Connect to Socket.IO server.
socket.connect();
</script>
```

Well we have the connection setup to receive events, but what about sending them? That's easy too.

```
<input type="text" id="msg" />
<input type="submit" onclick="send();" />
<script>
// Send message to server.
function send(){
    var msg = document.getElementById('msg');
    socket.send(msg.value);
    msg.value = '';
    return false;
}
</script>
```

Easy as π

Technologies Presented:

Node.js

<http://nodejs.org>

Express

<http://expressjs.com>

Socket.IO

<http://socket.io>

Questions?

Try it online at;
<http://chat.stephenbelanger.com>

Presented by Stephen Belanger