Node JS, Socket IO y la web en tiempo real

German Popoter

¿Quién soy?

@gpopoteur

8 años programando
3 profesionalmente

Desarrollo Web & Mobile
UI & UX Fanatic
Coffe & Pizza lover

¿Qué es Node Js?

- Node JS permite construir aplicaciones escalables en red usando JavaScript del lado del server.
- Node JS corre sobre Google Chrome's V8 JavaScript Runtine
- Node JS es muy veloz porque tanto Node JS como el V8 ***están escritos en su mayoria en C***
- V8 convierte JavaScript en código de maquina
- Fue creado en el 2009

¿Quién lo creó?

- Ryan Dahl (https://github.com/ry)

¿Qué **No** es Node Js?

- Multi-threaded
- Un framework web (No reemplazará a Rails, ni a ASP MVC)

¿Qué se puede hacer con Node Js?

- Servidor de Websockets
- Streaming
- Un cliente para subir múltiples archivos de manera rápida
- Cualquier aplicación que requiera data en tiempo real.

¿Por qué JavaScript?

- Es un lenguaje completo
- La mayoria de developers web ya lo conocen
- JavaScript está basado totalmente en eventos, no threads.

¿Cómo ayuda esto a Node?

- Mejor manejo de aplicaciones de alta concurrencia.
- Un evento no bloquea al próximo evento.

Blocking vs Non-Blocking

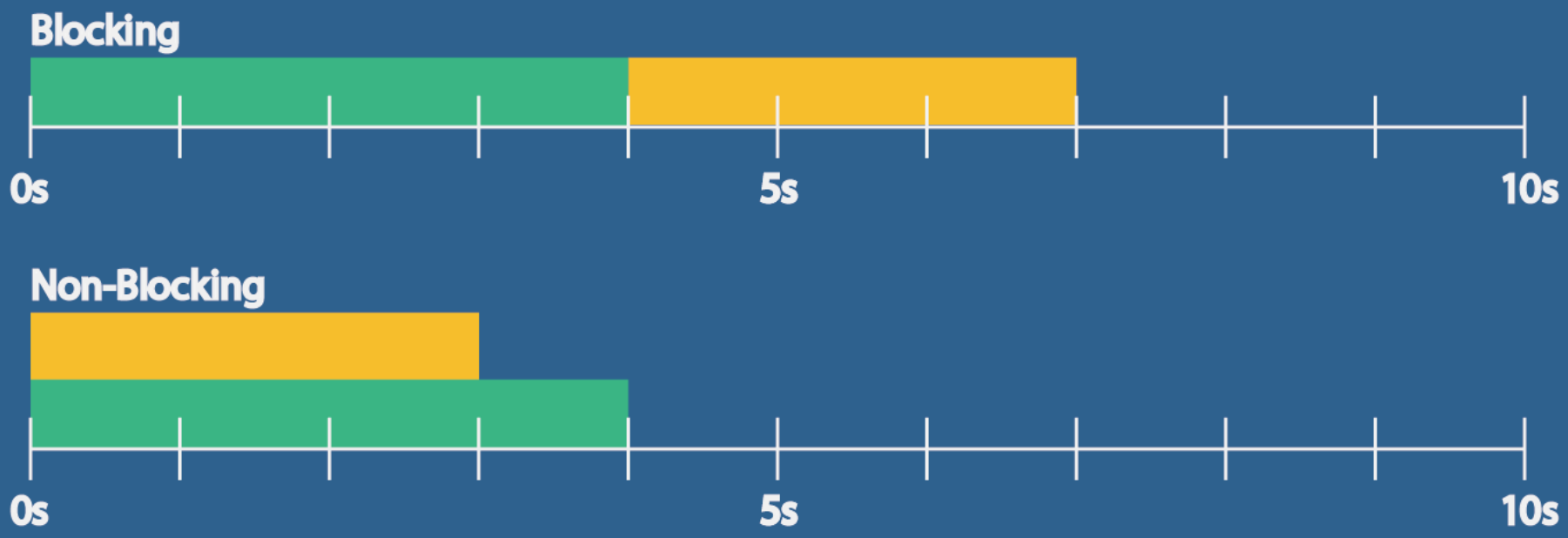
Blocking:

```
var contenido = fs.readFileSync('/dir');
console.log(contenido);
console.log('mas cosas...');
```

Non-Blocking:

```
fs.readFile('/dir',function (err, conteido) {
    console.log(contenido);
});
console.log('mas cosas...');
```

Blocking vs Non-Blocking



Ejemplos Blocking

Blocking común:

- Leer/Escribir en la DB
- Renderizando Website
- Llamando un WebService

Eventos

Eventos convencionales disparados por el DOM

- Click
- KeyUp
- Submit
- otros …

Node JS también dispara eventos

- Request
- Connection
- Data
- otros …

Eventos

```
// jQuery
$('#my-id').on('click', function(){ ... });

// Node JS
var callback = function(stream) {
  console.log('alguien se conecto! :)');
};
server.on('connection', callback);
```

Streams

“La transmisión continua de datos de una ubicación a otra. Por ejemplo, un video en tiempo real que se está descargando mientras se lo mira.”

Pueden ser de Lectura, Escritura o Ambos.

Hello World Time

```
var http = require('http');

http.createServer(function (request, response) {

    // Le dejamos saber al cliente OK
    response.writeHead(200, {"Content-Type": "text/plain"});
    // Escribimos Hello World
    response.end("Hello World\n");
}).listen(8000);
```

Streaming

```
var http = require('http');

http.createServer(function (request, response) {

    // Le dejamos saber al cliente OK
    response.writeHead(200, {"Content-Type": "text/plain"});

    // Lo que llegue por Request, lo envio por Response
    request.on('data',function (chunk) {
        response.write(chunk));
    };

    // Al terminar de recibir, le informo al cliente que termine
    request.on('end', function (){
        response.end();
    });

}).listen(8000);
```

Streaming

Problemas de este método:

- La lectura es mas rápida que la escritura
- Buffer lleno, detener la lectura, al vaciar buffer continuar

Solución:

.pipe(); (Parecido al comando Pipe en Unix)

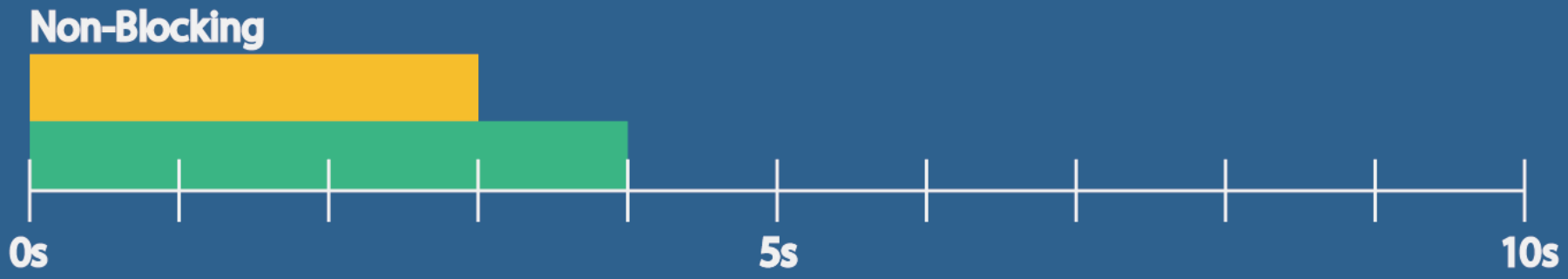
Streaming

```
var http = require('http');

var server = http.createServer(function (request, response) {

    // Le dejamos saber al cliente OK
    response.writeHead(200, {"Content-Type": "text/plain"});

    // Lo que llegue por Request, lo envio por Response
    request.pipe(response);
});

server.listen(8000);
```

Uploading File

```
var http = require('http');

var server = http.createServer(function (request, response) {

    // Le dejamos saber al cliente OK
    response.writeHead(200, {"Content-Type": "text/plain"});

    // Creo un archivo para Streaming
    var archivo = fs.createWriteStream("stream.txt");

    // Lo que llegue por Request, lo envio a un archivo
    request.pipe(archivo);

    // Al terminar de recibir, le informo al cliente que termine
    request.on('end', function (){
        response.end('Archivo Subiddo!');
    });
});

server.listen(8000);
```

Uploading



Varios archivos subiendo simultáneamente

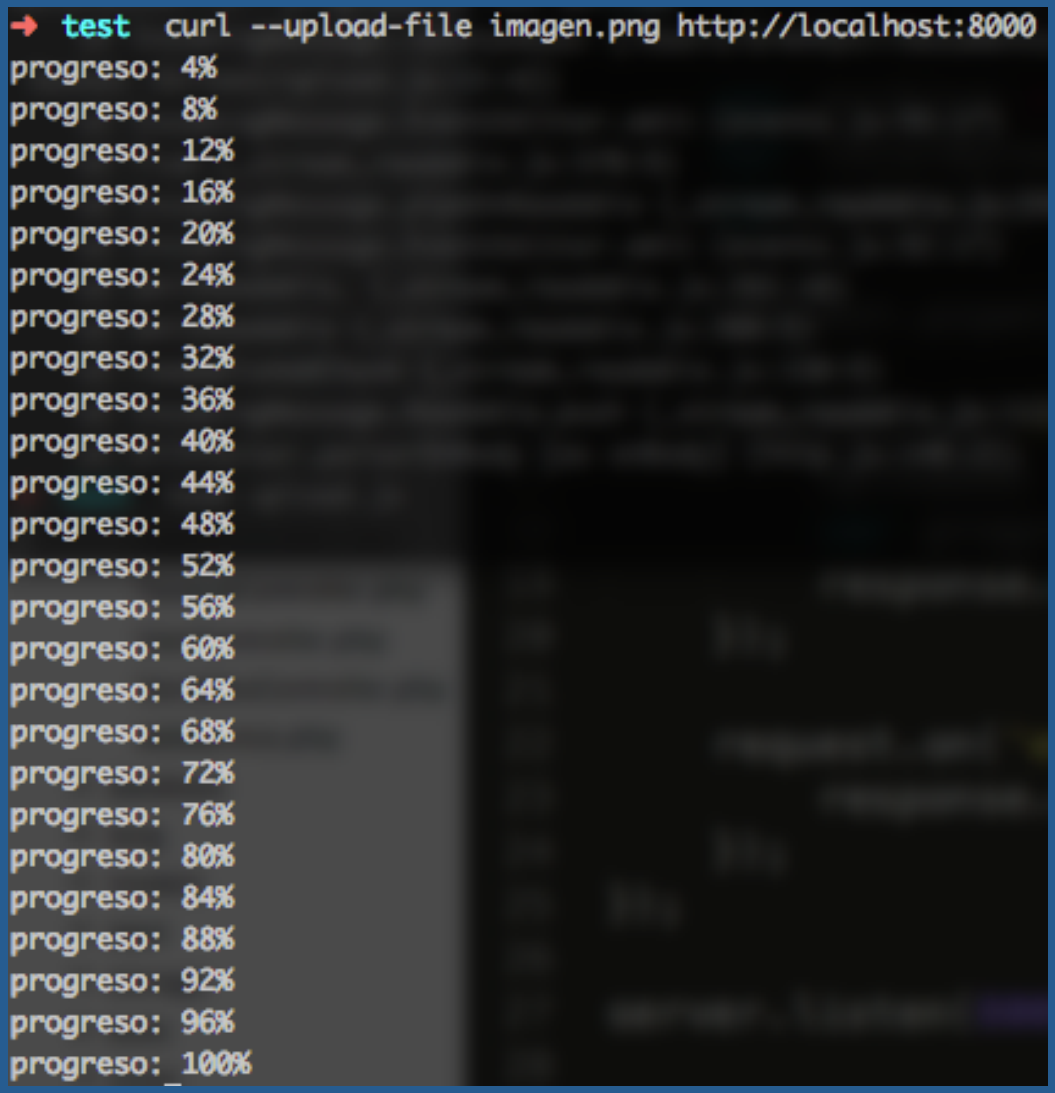
Uploading Progress

- Pain in the a**
- Difícil hacer en servidores comunes, ya que son blocking

Uploading Progress

```
var http = require('http');
var fs = require('fs')

var server = http.createServer(function (request, response) {

    var archivo = fs.createWriteStream("stream.txt");
    var totalBytes = request.headers['content-length'];
    var uploaded = 0;

    request.pipe(archivo);

    request.on('data', function (chunk) {
        uploaded += chunk.length;
        var progress = (uploaded / totalBytes) * 100;
        response.write("progreso: " + parseInt(progress, 10) + "%\n");
    });

    request.on('end', function (){
        response.end('Archivo Subido!');
    });
});

server.listen(8000);
```

Progreso


→ test curl --upload-file imagen.png http://localhost:8000
progreso: 4%
progreso: 8%
progreso: 12%
progreso: 16%
progreso: 20%
progreso: 24%
progreso: 28%
progreso: 32%
progreso: 36%
progreso: 40%
progreso: 44%
progreso: 48%
progreso: 52%
progreso: 56%
progreso: 60%
progreso: 64%
progreso: 68%
progreso: 72%
progreso: 76%
progreso: 80%
progreso: 84%
progreso: 88%
progreso: 92%
progreso: 96%
progreso: 100%

Módulos

- Carga módulo con require()
- Son archivos que contienen lógica
- Ayuda a organizar el proyecto
- Permite compartir código entre archivos

Crear un Módulo

```
// modulo mensajes.js
var saluda = function () {
    console.log("Hola!");
};
var despide = function (){
    console.log("Adios!");
};

exports.saluda = saluda;
exports.despide = despide;
```

```
var mensajes = require('./saluda');
mensajes.saluda();  // Hola!
mensajes.despide(); // Adios!
```

NPM

- Node Package Manager
- Manejo de Dependencias
- Instalación de Modules de 3ros
  - npm install socket.io

Módulos

- Express Js
- Socket IO
- Sails JS
- Request
- Underscore JS
- Jade
- Mongoose
- etc...

Socket.IO y Web Sockets

- Real-Time Web

Request Tradicional:

- Cliente envía request a server
- Server responde a cliente

Socket IO

Web Sockets

- Cliente y server siempre en comunicación

Socket IO

- Server

```
var io = require('socket.io').listen(8080);

io.sockets.on('connection', function (socket) {
  socket.emit('saluda', { mensaje: 'hello world' });
  socket.on('otro evento', function (data) {
    console.log(data);
  });
});
```

Socket IO

- Cliente

```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io.connect('http://localhost:8080');
  socket.on('saluda', function (data) {
    console.log(data);
    socket.emit('otro evento', { my: 'data' });
  });
</script>
```

Ejemplo de aplicación

¿Qué esperar de Node?

Gracias! :)

@gpopoteur

Powered by coffe