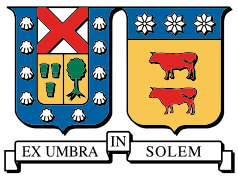


Redes de Computadores I

Informe de Proyecto

Node.JS: Plataforma de fácil programación de servidores para aplicaciones de red escalables

Profesor Agustín González

Integrantes Claudio Campusano
Nicolas Fredes
Felipe Cordero
Cristóbal Badilla

Fecha 28 de Julio de 2014

Versión 1.0

1. Resumen

En este trabajo se mostrará un entorno de programación sencilla que tiene varias utilidades, como es NODE.JS. Siendo específico, se mostrará el contenido visto en clases pero desde el punto de vista del servidor y no desde el punto de vista del cliente. Para ejemplificar, mostraremos la programación de un socket UDP, un socket TCP, la programación de encabezados y un servidor web.

2. Introducción

Node.js es un entorno de programación basado en el lenguaje de programación javascript, el cual actúa del lado del servidor, este cambia la noción de como se debe trabajar un servidor, fue creado con la meta de permitir al programador construir aplicación de red altamente escalables, esto quiere decir que el código puede soporta decenas de miles de conexiones simultaneas, lo cual no lo limita, (como por ejemplo los servidores web), esto resuelve el problema de los actuales lenguajes de programación los cuales se veían limitado la a cantidad de memoria RAM disponible para poder aceptar/iniciar una nueva conexión. En el transcurso de este informe demostraremos como trabajar con este entorno de programación, programando un par de aplicaciones básicas, como lo son servidores web, socket's tcp y udp y el armado de encabezados, ademas explicaremos como se lo logro y demostremos lo útil y sencillo que este entorno de programación llamado Node.js

3. Node.JS

Node.js es un entorno de programación del lado del servidor basado en el lenguaje de programación Javascript, que utiliza un modelo asíncrono, esto quiere decir que las entradas y salidas no bloquean al programa, usa una programación orientada a eventos con la particularidad que los eventos en esta plataforma son orientados a cosas que suceden del lado del servidor y no del lado del cliente como los que conocemos anteriormente en Javascript çomún". Está basado en el motor Javascript V8 un intérprete ultra rápido escrito en C++ (con el que funciona el javascript de Google chrome). Fue creado en 2009 por Ryan Dahl.

La meta primordial del Node.js es proporcionar una manera fácil para la construcción de programas de red escalables, este objetivo se da en vista de la actual situación que presentan los lenguajes de programación de hoy en día, donde la cantidad máxima de conexiones está limitada por la cantidad de memoria RAM que tenga el sistema, Node.js resuelve esto cambiando la forma en que se realiza la conexión, ya que en vez de hacer lo anteriormente descrito, dispara la ejecución de un evento dentro del proceso de motor de node.js.

En teoría Node puede soportar tantas conexiones como sockets tenga el sistema. En un sistema UNIX este límite puede rondar por las 65.000 conexiones, un número muy alto teóricamente. Sin embargo, en la realidad, la cifra depende de muchos factores, como la cantidad de información que la aplicación este distribuyendo a los clientes. A pesar de esto una aplicación podría mantener 20.000-25.000 clientes a la vez sin haber apenas retardo en las respuestas. Queremos recalcar que Node.js es un programa de servidor, este no viene a reemplazar a lo que son Apache o Tomcat que son básicamente servidores listos para instalar e implementar.

3.1. ¿Qué es un socket?

Un socket es la forma en que procesos en diferentes maquinas de una red se logran comunicar. Estos proporcionan un punto de comunicación entre los procesos mediante el cual se podrá, tanto, recibir como enviar información.

3.1.1. Socket UDP

Un socket udp, basa su programación en el protocolo UDP, el cual se caracteriza por ser "no orientado a la conexión", esto quiere decir decir que no se garantiza la llegada de los mensajes, ni su tiempo, osea que mediante un socket UDP un proceso puede ponerse a recibir o enviar información sin necesidad de comprobación alguna, de que "del otro extremo" se este "escuchando". Este tipo de socket se ocupan regularmente para información "no vital", como por ejemplo vídeos, chat's.

En este informe desarrollaremos socket UDP del lado del servidor el cual solo recibirá los mensajes y del lado del servidor mostrara los mensajes que le lleguen. A continuación una imagen con un ejemplo de un socket UDP programado usando Node.js.

```
1  var dgram = require("dgram");
2  var PORT = 12345;
3  var HOST = '192.168.50.10'
4
5  var sock = dgram.createSocket("udp4");
6
7  sock.on("error", function (err) {
8    console.log("Error en servidor:\n" + err.stack);
9    sock.close();
10 });
11
12 sock.on("message", function (msg, rinfo) {
13   console.log("El servidor ha recibido: " + msg + " de " + rinfo.address + ":" + rinfo.port);
14 });
15
16 sock.on("listening", function () {
17   var address = sock.address();
18   console.log("Servidor escuchando " + address.address + ":" + address.port);
19 });
20 sock.bind(PORT,HOST);
21
```

Con el comando "require", exportamos las bibliotecas que vamos a necesitar y las vamos asignando a variables, nuestro Socket UDP sera conocido con la variable sock, en la linea 5 especificamos el tipo de protocolo por el cual regirá nuestro socket.
El socket se encuentra "escuchando" en el puerto 12345 y desde una ip especificada, esto se le asigna mediante el comando sock.bin(Puerto,IP);
Este es un entorno que se basa en lenguaje de programación orientado a eventos por lo que a medida que se presenten estos, el programa responderá, veamos por ejemplo que ocurre cuando se presenta el evento "messaje". El programa en el caso de recibir un mensaje, mostrara por consola (usando el comando console.log()) toda la información referente a cual es el mensaje y desde que dirección y puerto vienen (todo esto solamente del lado del servidor).

3.1.2. Socket TCP

En este ejemplo se logró realizar una aplicación que realizara un eco al cliente. Para ello se ocupó el paquete `net`. En el código se pueden visualizar algunas partes importantes como lo son:

- net.createServer(function(sock)...); : En este lugar es donde creamos el socket
- .listen(PORT, HOST); : Acá el socket una vez creado se pone a la escucha en el puerto y host indicados. Si se detecta una conección la función dentro de net.createServer es accionada.
- sock.on('data', function(data)...); : Esta función se acciona si se reciben datos;
- sock.write : escribe los datos que se quieren mandar al cliente;
- sock.on('close', function(data)...); : Esta función se acciona si se cierra la conexión.

A continuación se muestra el código:

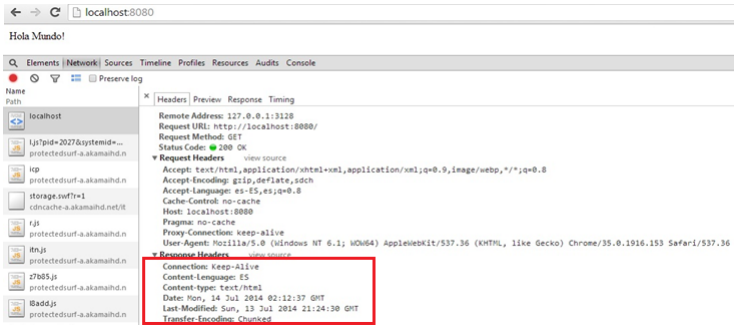
```
var net = require('net');

var HOST = 'localhost';
var PORT = 12345;
net.createServer(function(sock) {
    // We have a connection - a socket object is assigned to the connection
automatically
    console.log('CONNECTED: ' + sock.remoteAddress +':'+ sock.remotePort);
    // Add a 'data' event handler to this instance of socket
    sock.on('data', function(data) {
        console.log('DATA ' + sock.remoteAddress + ': ' + data);
        // Write the data back to the socket, the client will receive it as
data from the server
        sock.write('You said:' + data );
    });
    // Add a 'close' event handler to this instance of socket
    sock.on('close', function(data) {
        console.log('CLOSED: ' + sock.remoteAddress +' '+ sock.remotePort);
    });
}).listen(PORT, HOST);
console.log('Server listening on ' + HOST +':'+ PORT);
```

3.2. Encabezados

Para modificar los encabezados de la respuesta del servidor web, solo se necesita el uso del codigo "*resp.writeHead*" que permite colocar los encabezados que se deseen, en este caso mostraremos dos ejemplos donde se veran encabezados con distinta información.

Ejemplo 1:

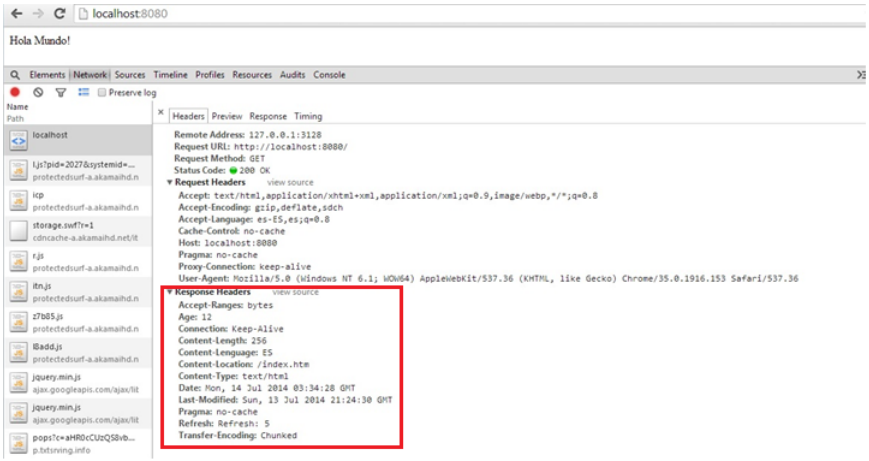
```
var http = require ('http');
var server = http.createServer();
function control (petic, resp) {
resp.writeHead(200, {
    'Content-type': 'text/html',
    'Content-Lenguage' : 'ES',
    'Last-Modified' : 'Sun, 13 Jul 2014 21:24:30 GMT',
    'Connection' : 'keep-alive',
    'Transfer-Encoding' : 'Chunked'

});

resp.write('Hola Mundo!');
resp.end();
 }
server.on('request', control);
server.listen (8080);
```



En este primer ejemplo, sólo se colocaron cuatro puntos como fue el lenguaje del contenido "*Content − Lenguage*", el tipo de contenido "*Content − type*", la última modificación "*Last − Modified*", el estado de conexión "*Connection*" y la codificación de transferencia "*Transfer − Encoding*".

Ejemplo 2:

```
1
2   var http = require ('http');
3   var server = http.createServer();
4   function control (petic, resp) {
5   resp.writeHead(200, {
6       'Accept-Ranges':'bytes',
7       'Age' : "12", //age the object has been in a proxy cache in seconds
8       'Connection' : 'keep-alive',
9       'Content-Type': 'text/html',
10      'Content-Length' : '256',
11      'Content-Lenguage' : 'ES',
12      'Content-Location' : '/index.htm',
13      'Last-Modified' : 'Sun, 13 Jul 2014 21:24:30 GMT',
14      'Pragma' : 'no-cache',
15      'Refresh' : 'Refresh: 5',
16      'Transfer-Encoding' : 'Chunked'
17
18  });
19
20  resp.write('Hola Mundo!');
21  resp.end();
22   }
23  server.on('request', control);
24  server.listen (8080);
```



En este segundo ejemplo, se colocarón los mismos puntos del ejemplo anterior pero se colocaron mas puntos para agrandar el encabezado, las informaciones que se agregarón son "$Accept - Range$", tiempo que lleva en el caché en segundos "Age", el largo del contenido "$Content - Lenght$", la ubicación del contenido "$Content - Location$", "$pragma$" y "$Refresh$".

3.3. Servidor Web

En este ejemplo se muestra como hacer un simple servidor web que tiene implementado el servicio `GET` y `POST`. Para lograr aquello fue necesario importar el paquete Express. El paquete Express, permite manejar el servicio REST de una manera más fácil que con el paquete http y es por esto que es uno de los más usados a la hora de programar servidores web. El ejemplo tiene dos partes, la primera parte es donde el servidor recibe un GET, el cual responde con un formulario HTTP simple (ver figura 1). Para ello en Node.JS se ocupa la función app.get(). Luego de que la persona completa el formulario, el explorador hace un POST al servidor con los datos del formulario. Para recibir el POST,en Node.Js se ocupa la función app.post(). El servidor recibe los datos y envía otra página web (Ver figura 2), devolviendo los datos escritos en la página web además de un applet escrito en java. En los anexos se encuentran los códigos del servidor HTTP y las captúras de pantallas.

4. Conclusiones

Finalmente pudimos notar, a través de los diferentes tópicos visto, como NODE.js nos presenta diferentes herramientas para poder programar en su plataforma de una manera limpia y fácil, aplicaciones de red, ademas de interiorizarnos de las ventajas de utilizar este tipo de plataforma y su cabida en aplicaciones de red futuras.

Por ultimo queríamos recalcar lo fundamental que fue realizar este trabajo ya que pusimos en juego la materia vista en clases, ademas de aprender/haciendo como realmente funcionan los socket's e servidores web, desde una perspectiva del servidor, lo que habitualmente se omite.

5. Anexos

5.1. Videos demostrativos

- Server UDP: https://www.youtube.com/watch?v=2HYSTYhwV-A
- Server TCP: http://youtu.be/ByL5-XdzpMU
- Server HTTP:

1.- http://youtu.be/rZ2Ag0XRolg

2.- http://youtu.be/TAzDoK-yaiY

5.2. Códigos para servidor HTTP

5.2.1. Código Node.js para servidor

```
1  var http    = require('http');
2  var express = require('express');
3  var bodyParser = require('body-parser');
4
5  var app = express();
6
7  app.use(bodyParser.urlencoded({
8    extended: true
9  }));
10
11 app.set('view engine', 'ejs');
12 app.set('views', __dirname + '/views');
13 app.use(bodyParser.json());
14
15 app.set('title', 'Hello World');
16 app.get('/', function  (req,res) {
17   console.log( "require: " + req);
18   res.sendfile('hello.html');
19
20 });
21
22 app.post('/',function (req, res) {
23   var info = {firstname: req.body.FirstName, lastname:req.body.LastName};
24   res.render('response',{firstname: req.body.FirstName,
25     lastname:req.body.LastName});
26   console.log(info);
27 });
28
29 app.listen('5000');
```

5.2.2. Código HTML para primera respuesta del servidor a petición GET hecha por el cliente

```
1
2  <!DOCTYPE html>
3  <html>
4
5  <head>
6    <title>My first page in express </title>
7  </head>
8
9  <body>
10
11   <h1>Hello World!</h1>
12
13   <p>This is my first paragraph</p>
14
15   <p>This is a POST
16     <p>
17     <form action="http://localhost:5000" method="post">
18         First name: <input type="text" name="FirstName" placeholder = "First Name"
               ><br>
19         Last name: <input type="text" name="LastName" placeholder = "Last Name" ><
               br>
20         <input type="submit" value="send">
21       </form>
22
23     </p>
24
25
26   </p>
27
28 </body>
29
30 </html>
```

5.2.3. Código HTML para la respuesta del servidor debido a un POST hecho por el cliente

```
1
2  <head>
3    <title>
4      Response
5    </title>
6  </head>
7
8  <h1>This is a responce for your Request</h1>
9
10 <p>
11   Hello <b> <%=firstname%> <%=lastname%></b>
12   Are you ready for use this java applet?
13 </p>
14
15 <applet archive="PhysicsLab.jar" code=PhysicsLabApplet.class width="1300" height="
       500" >
16     <param name="title" value="Physics Lab: 3 Experiencing with oscillations"/>
17     <param name="fixedHookNum" value="2"/>
18    <!-- <param name="fixedHook.x" value="position"/> -->
19     <param name="fixedHook.1" value="0.5"/>
20     <param name="fixedHook.2" value="1.5"/>
21     <param name="ballNum" value="3"/>
22 <!--  <param name="ball.x" value="speed; mass; pos_x;radius "/>  -->
23     <param name="ball.1" value="0; 1; 1; 0.1"/>
24     <param name="ball.2" value="0; 1; 2; 0.1"/>
25     <param name="ball.3" value="0; 1; 3; 0.1"/>
26     <param name="oscillatorNum" value="1"/>
27    <!-- <param name="oscillator.x" value="amplitude; frequency; position"/> -->
28     <param name="oscillator.1" value="0.5; 1; 2.5"/>
29     <param name="deltaTime" value="0.001"/>
30     <param name="refreshTime" value="0.005"/>
31     <param name="maxPlotTime" value="30"/>
32   </applet>
33
34 <form action="http://localhost:5000/" method="get">
35 <button type="submit">Reload</button>
36 </form>
```

5.3. Imágenes de respuestas del servidor HTTP

Figura 1: Primera respuesta a GET enviado hacia el servidor

Figura 2: Imagen a respuesta de POST recibido por el servidor

5.4. Bibliografía

- http://nodejs.org/
- http://en.wikipedia.org/wiki/Node.js
- https://medium.com/@edwardoregan/why-node-js-matters-bb49dbf688af
- http://devopsangle.com/2013/04/01/the-birth-of-node-where-did-it-come-from-creator-ryan-dahl-shares-the-history/
- http://stackoverflow.com/questions/5062614/how-to-decide-when-to-use-node-js
- http://dsp.mx/blog/sistemas-de-informacion/49-sockets-tcp-udp