Developing Web services using Node JS

Welcome to Gemini Tech Blog! In today's blog we shall explore on how Node js can be used for developing web services.

Node js is relatively new when compared to its peers like PHP, .NET and Java. It has some subtle advantages over rest of the technologies which makes it quite popular and worth while to explore and implement wherever suited.

Before we go through some examples and cookbook instructions to build up a Simple Web service, it is quite important to try and understand the answer to the following question

---

What is Node js? Exactly how and when should it be used ?

NodeJs

It is an open-source, cross-platform runtime environment for developing server-side web applications. Its applications are written using JavaScript and can be run within the Node.js runtime environment on most of the platforms. It uses Google V8 JavaScript engine to execute code and does not require Web server software like Apache HTTP server, Nginx, IIS etc.

Advantages:

- Node.js operates on a single thread, using non-blocking I/O calls, allowing it to support tens of thousands of concurrent connections without incurring the cost of thread context switching.
- The design of sharing a single thread between all the requests is intended for building highly concurrent applications, where any function performing I/O must use a callback.

```
Example:
var fs = require('fs');  // A library for reading files
...
=== Request scope begins ===

    fs.readFile('ReadMe.txt', function(err, data){
    // Callback function called on completion of file reading task
            if(err){
                // report error
            }
            // Do something with data
   });

=== Request scope ends ===
...
```

- A single thread is responsible for executing the readFile function and its callback. This increases the throughput of I/O bound operations.
- Developers can create highly scalable servers without using threading, by using a simplified model of event-driven programming that uses callbacks to signal the completion of a task. Concurrency is difficult in many server-side programming languages, and can lead to poor performance if proper trade off is not met.

Where should it be used ?

There are certain pain points in older technologies which are easily mitigated using Nodejs. It would be best to use it in those areas as there is no better replacement than Node js itself.

Following are some of them:

- When there is a need to use a rich client framework (Angular, Ember, Backbone) and a REST-ful server-side API that shuttles JSON back and forth. Even if such frameworks are not used and some custom jQuery code is written, there will be a constant need for translating that data into some relevant classes or other data structures. Node Js gives the capability of writing web services directly in JavaScript. This enables the developers to do development quickly and efficiently.
- When there is a need to use a NoSQL DB. If an object database like Mongo is to be used, then Javascript can be easily extended to its persistence layer. So the same language on the client, on the server, and in the database interaction layer can be used. Reducing development effort.
- When there is a need to build a web service that has to scale well enough to serve millions of users without much dip in performance. Visit this link to have a look for performance comparision of Nodejs against Apache + PHP.

Tools needed to start off

For the purpose of simplicity and uniformity, we shall go for certain platforms and frameworks over others. These choices shall help in longer run as we get more and more familiar.

Requirements: Any machine having Linux like OS.

Install Curl client on your machine

curl download

or if your machine is debian based then simply do

sudo apt-get install curl

Install Nodejs

nodejs download

on debian based machines

sudo apt-get install nodejs

And we are done with configurations!

Baby steps

Before creating a full blown HTTP server, we need to learn how to create a simple program in nodejs.

Try this:

- Create a file **helloworld.js** using vim, nano or any other editor you like.`console.log('Hello World');`
- Exit editor and run the file using following command `node helloworld.js`
  - Done !? Good enough. But this certainly does not solve your requirement of creating flexible and scalable web applications. So moving ahead...

Some modifications:

- Let us create a simple text file in the current working directory as **index.html** having following contents: `<html> <body><p>Hello World</p></body> </html>`
- Now add a function that reads the text file kept in current directory.

- Here is the code for it:

```
console.log('Hello World');

var fs = require('fs'); // A lib that reads files

fs.readFile('index.html', function(err, data){
    // callback function called when file is read.

    // report and return if error occurs.
    if(err) { console.log(err); return; }

    // print file contents
    console.log(data.toString());
});
```

- Execute the same command: **node helloworld.js** All good till now... You should see the contents of the file.
- Next we shall plan to serve this index.html page on client's browser. For that we would need to create an HTTP server that listens to client's requests and serves the page.

A very simple HTTP server:

To create a simple HTTP server we would need to use following library

http

Following is how we use it:

```
var http = require('http');
var server = http.createServer(function(request, response){

    response.writeHead(200, {'Content-Type':'text/html'});
    fs.readFile('index.html', function(err, data){
            if(err){
                    console.log(err);
                    return;
            }

            response.write(data.toString());
            response.end();
            console.log('Response sent to client')
    });
```

```
});
server.listen(8080, 'localhost');
```

Next lets open the browser and hit the following URL: http://localhost:8080 contents of index.html shall be displayed.

As for now the server we created can serve static HTML pages. Now lets try transforming the server so that it can exchange JSON data. We shall find how easy it is to do.

A simple HTTP-JSON-API server

Create a new file as http-json-server.js and add the following code:

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

var server = http.createServer(function(request, response){
        var clientName;
        request.on('data', function(data){
                var info = JSON.parse(data.toString());
                console.log('Client sent: %j', info);
                clientName = info.name;
        });
        request.on('end', function(){
                console.log('Client rquest ended');
                response.writeHead(200, {'Content-Type': 'application/json'});
                response.write(JSON.stringify({hello: clientName}));
                response.end();
                console.log('Response sent');
        });

});

server.listen(8085, 'localhost');
```

Now lets discuss the behavior of this code and how we can use this server for exchanging JSON data. 1. The statement *request.on('data', function...)* has registered for a callback that is executed every time some data chunk is received from client. Javascript being event driven in nature, works by attaching event listeners (callbacks) to events being fired. 'data' is one such name of that event. More about that later. 2. Similarly, *request.on('end', function..)* is registered to an event when client finishes off sending it's request. 3. What we need to ensure on the server's end is that we are sending the request only when the client has finished off requesting. That is because Node js is asynchronous by nature, the lines, *request.on('data', function...)* etc are just **registering the listeners** and **not actually executing** them (very counter intuitive at first, right...) but this what makes Node js incredibly fast! 4. Hence, the lines of code *response.writeHead(200, ..) response.write(/* some data */); response.end();* Need to be inside the callback of

request.on('end',function..)

Now lets start the server:

```
node http-json-server.js
```

Test

Execute following command to test curl -i http://localhost:8085 -X POST -d '{"name": "YourName"}'

> **Note**: If you get an error of the type curl: (3) [globbing] unmatched close brace/bracket in column 15 That is because the double quotes and single quotes copied from this source are not proper (JSON becomes invalid) and you need to type the command by hand.

You should get following response:

```
{"hello":"YourName"}
```