TCP Socket Programming in Node.js

Posted on October 26th, 2011 under Node.js
Tags: Client, node.js, Server, Socket, TCP

Programming TCP Sockets in Node.js

Eager to know how sockets are programmed in Node? There are three variants of sockets in Node - i. TCP, ii. UDP, iii. UNIX domain. In this particular post, I will show you the basics of TCP socket programming in Node.js.

There are two categories of TCP socket programs you can write - i. server, ii. client. A TCP server listens for connections to it from clients and send data to the client. A TCP client connects to a TCP server exchange data with it. The communication between client and server happens via sockets.

Programming TCP sockets in Node requires the `net` module, which is an asynchronous wrapper for network programming. The `net` module is capable of many things, but for today we'll just focus on creating a TCP server and a client.

Writing a TCP Server

Here is an example of a very simple TCP server written in Node. Read the comments thoroughly, it explains how the code works.

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

var HOST = '127.0.0.1';
var PORT = 6969;

// Create a server instance, and chain the listen function to it
// The function passed to net.createServer() becomes the event handler for the
// The sock object the callback function receives UNIQUE for each connection
net.createServer(function(sock) {

    // We have a connection - a socket object is assigned to the connection auto
    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
        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);
```

The same thing can be accomplished in a slightly different way. Make sure to include the necessary variables from the last example for this one to work.

```
var server = net.createServer();
server.listen(PORT, HOST);
console.log('Server listening on ' + server.address().address +':'+ server.addre
server.on('connection', function(sock) {

    console.log('CONNECTED: ' + sock.remoteAddress +':'+ sock.remotePort);
    // other stuff is the same from here

});
```

So what's the difference? Basically they are the same thing, it's just we used different conventions of the JavaScript language. In the first example, we passed the `connection` event handler to `net.createServer()`, and chained the `listen()` function. In the latter, we took a more 'conventional' approach. Either way works.

Writing a TCP Client

Now let's write a client to connect to the server we created. The following code creates a simple client which connects to the server, sends a message to server, and disconnects after getting a response from the server. Read the comments to follow the code.

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

var HOST = '127.0.0.1';
var PORT = 6969;

var client = new net.Socket();
client.connect(PORT, HOST, function() {

    console.log('CONNECTED TO: ' + HOST + ':' + PORT);
    // Write a message to the socket as soon as the client is connected, the se
```

```
    client.write('I am Chuck Norris!');

});

// Add a 'data' event handler for the client socket
// data is what the server sent to this socket
client.on('data', function(data) {

    console.log('DATA: ' + data);
    // Close the client socket completely
    client.destroy();

});

// Add a 'close' event handler for the client socket
client.on('close', function() {
    console.log('Connection closed');
});
```

So that's the very basics of TCP socket programming in Node.js, hope it helped you understand socket programming in Node better. Note that socket programming is a lot more than these simple examples. Once you start exchanging huge chunks of data and want to do complex things you will need to understand and use Streams and Buffers among other things.

Further Reading

1. Node.js net Module
2. Node.js Streams
3. Node.js Buffers

Node.js UDP Server and Client Example Using Node.js to download files

Using MySQL with Node.js	Using Redis with Node.js
Node.js Async Programming	jQuery with Node.js
How to install Node.js on Webfaction	Node.js HTTPS – SSL Certificate
Node.js Restart on File Change	Node.js Tutorial
Understanding directory references in Node.js	Scripting a Node.js App

Related to this post

i. Express.js HTTPS Server Client Example
ii. Node.js UDP Server and Client Example
iii. Using Node.js to download files
iv. A Port Scanner in Node.js
v. Node.js EventEmitter Tutorial
vi. Using MySQL with Node.js
vii. Using Redis with Node.js
viii. jQuery with Node.js
ix. How to install Node.js on Webfaction
x. vhost in Express.js

17 Responses to “TCP Socket Programming in Node.js”

Preetam says:
January 14, 2013 at 10:43 pm
Firstly thanks for the response.

A link to the main js file called by node.exe
https://www.dropbox.com/s/15yvuqrp9418gg9/NodejsCode.txt

Please note:

* I have commented sections and brought the code down to the bare minimum that still causes an increase in the memory footprint.

* Also to let you know the TCP server is a C# application that is listening on port 60101 and when a client connects to it.. it sends some data every 250 milliseconds as long as the client is connected.

* I also added a console.log in the ‘data’ event that prints timestamp and I did get 4 timestamps printed every second.. my intention here was to make sure that the data was not queueing up causing the increase in the memory.

I have installed nodetime on the server and will continue working on the cause of the problem.. but any input from your end as to what I am doing wrong in the code will be much appreciated.

thanks heaps again for your time.

Regards,
Preetam

massi says:
January 29, 2013 at 4:33 pm
I’ve tried this tutorial and I discovered that

server.address()

return null if called outside the connection callback. So in the second example I can't log onto console

server.address().address

cause it return "undefined". However the same code inside the serverCreate() callback works.
Anyone know why?

By the way, this is a very useful tutorial!


chris says:
May 31, 2014 at 2:35 pm
Thank you for a very interesting and useful article.

However this is the first time I have come across node.js however I have done some js coding along with php and c++.

What I wanted to ask you is the following?

Could I write some js code (like your example) which would execute in the browser, get the client ip, and then open a socket and write it back to an external server?

Thanks in advance.


Yogesh says:
October 13, 2014 at 5:16 am
Hi,
Thanks for your valuable explanation regarding node.js.
I'm a newbee to node.js and in the beginning stages of learning node.js.

Could you please explain with examples on how to create Persistent Socket Connection between a TCP client and TCP Server. I've searched in the Internet and couldn't find any example.

Right now, I'm using the above TCP Client code to connect to a remote process which creates new PID every time it connects.

```
var client = new net.Socket();
client.connect(PORT, HOST, function() {
console.log('CONNECTED TO: ' + HOST + ':' + PORT);
// Write a message to the socket as soon as the client is connected, the server will receive it as message from the client
client.write('I am Chuck Norris!');
});
```

Where as, what I want to achieve is, using the same PID between various messages sent from TCP Client (node.js) to TCP Server. So, I want to know how to create a Persistent Socket connection in node.js.

Thanks again for your efforts.

Thanks,
Yogesh

 deepak koirala says:
February 25, 2016 at 11:08 am
thank you very much

 eliot says:
December 23, 2016 at 12:45 am
Hi I am new to Node js. I am getting this error i have tried googling but never helps can anyone help me with this error.

Uncaught Error: Module name "net" has not been loaded yet for context: _. Use require([])

Can someone give me a hand

Thanks

 Captain says:
December 23, 2016 at 4:33 am

Hard to tell what's going on without looking at the whole code. I'd recommend posting the question on StackOverflow.

« Previous 1 2

Make a Comment

Your Name

Your E-Mail

Submit Comment

Follow @hacksparrow

Recent Comments

Captain on JavaScript .bind() vs .apply() and .call()
Kevin Kim on JavaScript .bind() vs .apply() and .call()
tiagojdferreira on Using Node.js to download files
Pedro Vagner on Difference between spawn and exec of Node.js child_process
Conor Doyle on The For Loop vs the For Each Loop in JavaScript
Jennifer on Node.js Module – exports vs module.exports
Gordon on Python: difference between list and tuple
kdmrobot on Mongoskin Tutorial with Examples
Captain on TCP Socket Programming in Node.js
eliot on TCP Socket Programming in Node.js

Tags

Apply Aptana array Call closure CURL database Error Express Express.js Firefox git GitHub Google Gzip Homebrew HTML HTTP HTTPS JavaScript javascript array javascript for loop Linux list lol Mac Microsoft MongoDB Mozilla MySQL mysqldump node.js NoSQL npm Object pagination PHP Python Redis Regex Sitemap string tuple Ubuntu Wordpress