



PEOPLE ALSO LIKE

- A Simple CRUD with MEAN Stack (MongoDB, ExpressJS, AngularJS, Node.js) + Sails.js on Windows
- Deploying Smart Contract and Node.js App [Full Stack Ethereum Dapp Part-5]
- Client Side Validation using ASP.NET Validator Controls from Javascript
- nvm-windows: Node or NPM not recognized after nvm installed
- Node.js: Sinon Stub and Passport Authenticator

Node.js: Simple TCP Server & Client and Promisify the Client

In this post, you will see an example of simple TCP server and client in traditional javascript way and in ES6 way. **Node.js** has **net** module which provides an asynchronous network API for creating stream-based TCP or IPC servers and clients. We are going to use it to implement TCP server and client. This post is updated with Node v6.11.2.

TCP Server:

Here is an example of TCP server in **Node.js**:

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

// Configuration parameters
var HOST = 'localhost';
var PORT = 1234;

// Create Server instance
var server = net.createServer(onClientConnected);

server.listen(PORT, HOST, function() {
  console.log(`server listening on %j`, server.address());
});

function onClientConnected(sock) {
  var remoteAddress = sock.remoteAddress + ':' + sock.remotePort;
  console.log(`new client connected: %s`, remoteAddress);

  sock.on('data', function(data) {
    console.log(`%s Says: %s`, remoteAddress, data);
    sock.write(data);
    sock.write('exit');
  });
  sock.on('close', function () {
    console.log(`connection from %s closed`, remoteAddress);
  });
  sock.on('error', function (err) {
    console.log(`Connection %s error: %s`, remoteAddress, err.message);
  });
}
```

TCP Client:

```
var net = require('net');
var HOST = 'localhost';
var PORT = 1234;
var client = new net.Socket();

client.connect(PORT, HOST, function() {
  console.log(`Client connected to: ` + HOST + `:` + PORT);
  // Write a message to the socket as soon as the client is connected, the server will
  client.write('Hello World!');
});

client.on('data', function(data) {
  console.log(`Client received: ` + data);
  if (data.toString().endsWith('exit')) {
    client.destroy();
  }
});

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

client.on('error', function(err) {
  console.error(err);
});
```

D:\node>node server.js
server listening on {"address":"127.0.0.1","family":"IPv4","port":1234}
new client connected: 127.0.0.1:62682
127.0.0.1:62682 Says: Hello World!
connection from 127.0.0.1:62682 closed

D:\node>node client.js
Client connected to: localhost:1234
Client received: Hello World!
Client received: exit
Client closed

ES6 Way:

Let's update the code in ES6 way. It is assumed you are familiar with following ES6 features:

- Class
- Arrow Functions
- Template literals

TCP Server:

```
'use strict';

// load the Node.js TCP library
const net = require('net');
const PORT = 1234;
const HOST = 'localhost';

class Server {
  constructor(port, address) {
    this.port = port || PORT;
    this.address = address || HOST;
    this.init();
  }

  init() {
    let server = this;
    let onClientConnected = (sock) => {

      let clientName = `${sock.remoteAddress}:${sock.remotePort}`;
      console.log(`new client connected: ${clientName}`);

      sock.on('data', (data) => {
        console.log(`${clientName} Says: ${data}`);
        sock.write(data);
        sock.write('exit');
      });

      sock.on('close', () => {
        console.log(`connection from ${clientName} closed`);
      });

      sock.on('error', (err) => {
        console.log(`Connection ${clientName} error: ${err.message}`);
      });
    };

    server.connection = net.createServer(onClientConnected);

    server.connection.listen(PORT, HOST, function() {
      console.log(`Server started at: ${HOST}:${PORT}`);
    });
  }
}

module.exports = Server;
```

To test server, use following code in another file and run it

```
const Server = require('./server');
new Server();
```

TCP Client:

```
'use strict';

const net = require('net');
const PORT = 1234;
const HOST = 'localhost';

class Client {
  constructor(port, address) {
    this.socket = new net.Socket();
    this.address = address || HOST;
    this.port = port || PORT;
    this.init();
  }

  init() {
    var client = this;

    client.socket.connect(client.port, client.address, () => {
      console.log(`Client connected to: ${client.address} : ${client.port}`);
      client.socket.write('Hello World!');
    });

    client.socket.on('data', (data) => {
      console.log(`Client received: ${data}`);
      if (data.toString().endsWith('exit')) {
        client.socket.destroy();
      }
    });

    client.socket.on('close', () => {
      console.log(`Client closed`);
    });

    client.socket.on('error', (err) => {
      console.error(err);
    });
  }
}

module.exports = Client;
```

To test client, use following code in another file and run it

```
const Client =require('./client');
new Client();
```

Output:

D:\node>node serverTest.js
Server started at: localhost:1234
new client connected: 127.0.0.1:62846
127.0.0.1:62846 Says: Hello World!
connection from 127.0.0.1:62846 closed

D:\node>node clientTest.js
Client connected to: localhost : 1234
Client received: Hello World!
Client received: exit
Client closed

Promisify the Client:

In your Node application, you might need to trigger server on a particular event like on button click and you want to get ES6 Promise object for neat implementation.

See Also: **JavaScript Promises: Understanding Error Handling with Example**

Let's create a method to write socket on client side and returns Promise object.

```
'use strict';

const net = require('net');
const PORT = 1234;
const HOST = 'localhost';

class Client {
  constructor(port, address) {
    this.socket = new net.Socket();
    this.address = address || HOST;
    this.port = port || PORT;
    this.init();
  }

  init() {
    var client = this;
    client.socket.connect(client.port, client.address, () => {
      console.log(`Client connected to: ${client.address} : ${client.port}`);
    });

    client.socket.on('close', () => {
      console.log(`Client closed`);
    });

    sendMessage(message) {
      var client = this;
      return new Promise((resolve, reject) => {

        client.socket.write(message);

        client.socket.on('data', (data) => {
          resolve(data);
          if (data.toString().endsWith('exit')) {
            client.socket.destroy();
          }
        });

        client.socket.on('error', (err) => {
          reject(err);
        });
      });
    }
  }
}

module.exports = Client;
```

sendMessage method returns **Promise** object. Let's see an example how to use it:

```
const Client =require('./client');
const client = new Client();

client.sendMessage('A')
.then((data)>=> { console.log('Received: ${data}'); return client.sendMessage('B');} )
.then((data)>=> { console.log('Received: ${data}'); return client.sendMessage('C');} )
.then((data)>=> { console.log('Received: ${data}'); return client.sendMessage('exit');} )
.catch((err) =>{ console.error(err); })
```

To test, remove following line in Server code

```
sock.write('exit');
```

Now **exit** is done by our code.

Output:

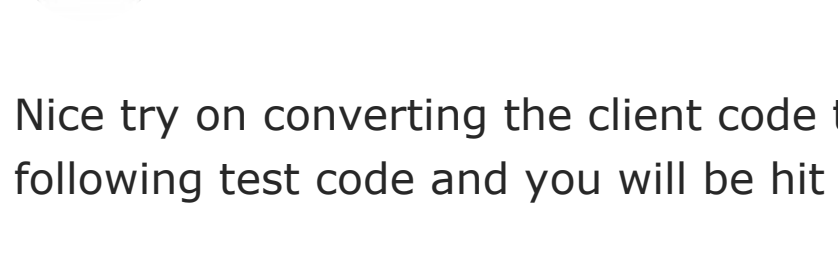
D:\node>node serverTest.js
Server started at: localhost:1234
new client connected: 127.0.0.1:63007
127.0.0.1:63007 Says: A
127.0.0.1:63007 Says: B
127.0.0.1:63007 Says: C
127.0.0.1:63007 Says: exit
connection from 127.0.0.1:63007 closed

D:\node>node clientTest.js
Client connected to: localhost : 1234
Received: A
Received: B
Received: C
Client closed

Conclusion:

This post has an example of TCP server and client in traditional javascript and ES6 way. Also, an example of promisify TCP client for better architecture.

Enjoy **Node.js** and ES6 !!



2 comments

midnightcodr
December 26, 2017 at 1:45 am

Nice try on converting the client code to promise but your version is not scalable, try to run the following test code and you will be hit with a MaxListenersExceededWarning.

```
const bluebird = require('bluebird')

const Client =require('./tc');

const client = new Client();

bluebird.each(Array(20).fill(0), n => {

  console.log('to send')

  return client.sendMessage('A')

}).then(() => {

  client.sendMessage('exit')

})
```

Someone
May 19, 2019 at 12:39 am

Hello midnightcodr

You need to add removeListener, because each time sendMessage is called, it adds new listeners. This is why it not scale.

```
socket.on('data', function getData (data) {
  this.removeListener('data', getData)
  return resolve(data)
})
```

This logic should be applied to other listeners as well.

By the way destroying a socket when there is no error is bad practice and not recommended.

socket.on('data') should also be able to handle chunked data.

HTH

REPLY

Leave a Reply

Your email address will not be published. Required fields are marked *

COMMENT

NAME *

EMAIL *

WEBSITE

POST COMMENT

Follow Us

2100+ fans

500+ subscribers

200+ followers

250+ followers

Subscribe via Email

Get updates when new tutorials are published on TechBrij. Enter your email to enroll.

Your Email

SUBSCRIBE

Categories

TensorFlow	Node.js
JavaScript	Python
The Best WordPress Plugins	ASP.NET, ASP.NET MVC, ASP.NET Core
AngularJS	Blogging
HTML,CSS	ASP.NET MVC Cheat Sheet
Basic Ubuntu Linux Commands	Projects
Cheat-sheets (Notes)	Online Tools

Recent Comments

- Santosh Sawant** on [WordPress: Disable 404 Redirect to Homepage](#)
Thanks for this article; it has very useful inform
- Pesticide Filler** on [Database Change Notifications in ASP.NET using SignalR and SignalDependency](#)
Great work! This is the type of information that s
- Ihsan Berahim** on [nvm-windows: Node or NPM not recognized after nvm installed](#)
This is correct. I just to add on for Powershell u
- aneru** on [Google Adsense Tips: My Adsense Placement Experiments](#)
Thanks for the post.
- Roger on RabbitVCS: A TortoiseGit for Ubuntu**
\$ yum install rabbitvcs-nautilus
- sushant d deshmukh** on [TensorFlow 2: Convolutional Neural Networks \(CNN\) and Image Classification](#)
Instead of mnist dataset take other dataset which