МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное

учреждение высшего профессионального образования

«Ярославский государственный университет им. П.Г.Демидова»

(ЯрГУ)

Кафедра компьютерной безопасности и

математических методов обработки информации

Курсовая работа

Сравнение производительности web-серверов, написанных на языках node js и C#

Научный руководитель

\_\_\_\_\_\_\_\_\_\_\_ О.В.Власова

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2016 г.

Студент группы КБ-41

\_\_\_\_\_\_\_\_\_\_\_\_ С.А.Попов

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2016 г.

Ярославль 2016

Содержание

# Введение

Рисунок 1

Рисунок 2

Рисунок 3

Рисунок 4

Рисунок 5

**Рисунок 6**

**Рисунок 7**

# Список литературы

1. Хэррон Д. "Node.js. Разработка серверных веб-приложений в JavaScript" - М.: ДМК Пресс, 2012. – 144с.: ил.
2. Сухов К.К. "Node.js. Путеводитель по технологии" - М.: ДМК Пресс, 2015. – 416с.: ил.
3. Кантелон М., Хартер М., Головайчук T., Райлих Н. "Node.js в действии". — СПб.: Питер, 2014. - 548с.: ил
4. Пауэрс Ш. "Изучаем Node.js" - СПБ.: Питер, 2014 - 400с.: ил.
5. Documentation : https://nodejs.org/. -URL: https://nodejs.org/dist/latest-v6.x/docs/api/
6. http://stackoverflow.com/. -URL: http://stackoverflow.com/questions/9290160/node-js-vs-net-performance

# Приложение

**Листинг 1**

udp-клиент на языке node js

'use strict'

const dgram = require("dgram")

const client = dgram.createSocket("udp4")

const options = {

portForReceiving: 3333,

portForSending: 3334,

iterations: 2048,

sizeOfMessage: 32768,

host: "127.0.0.1"

}

const message = Buffer.alloc(options.sizeOfMessage, '1', 'utf-8')

const sendMessage = (message) => client.send(message, 0, message.length, options.portForSending, options.host)

let time,

i = 0,

totalLength = 0

client.on("message", (msg) => {

totalLength += msg.length

i++

sendMessage(message)

i > options.iterations && client.close()

})

client.on('close', () => {

var [begin, start] = process.hrtime(time)

console.log(`total: ${totalLength}`)

console.log(`work time - ${(begin \* 1e9 + start) / 1000000} ms`)

console.log('Connection was closed for client')

process.exit()

})

client.bind(options.portForReceiving, options.host)

time = process.hrtime()

sendMessage(new Buffer('hello'))

**Листинг 2**

udp-сервер на языке node js

'use strict'

const dgram = require('dgram');

const server = dgram.createSocket('udp4');

const options = {

portForReceiving: 3334,

host: "127.0.0.1",

iterations: 2048

}

const sendMessage = (message, info) => server.send(message, 0, message.length, info.port, info.address);

let lengthContent = 0, iterationNumber = 0

server.on('close', () => console.log('Connection was closed for server'))

server.on('message', (message, info) => {

lengthContent += message.length

iterationNumber++

sendMessage(message,info)

if (iterationNumber > options.iterations) {

console.log(`total: ${lengthContent}`)

lengthContent = 0

iterationNumber = 0

}

})

server.on('error', (err) => {

server.close()

throw err;

})

server.bind(options.portForReceiving, options.host);

**Листинг 3**

tcp-клиент на языке node js

'use strict'

const net = require('net');

const client = new net.Socket();

const options = {

sizeOfMessage: 32768,

iterations: 2048,

port: 1337,

host: '127.0.0.1'

}

const message = Buffer.alloc(options.sizeOfMessage, '1', 'utf-8');

let contentLength = 0,

iterationNumber = 0,

time

client.on('readable', () => {

let chunk;

while (null !== (chunk = client.read())) {

contentLength += chunk.length

}

iterationNumber++

iterationNumber <= options.iterations && client.write(message)

})

client.on('end', () => {

let [begin,end] = process.hrtime(time)

console.log(`total client: ${contentLength}`)

console.log(`work time - ${(begin \* 1e9 + end) / 1000000} ms`)

console.log('Connection was closed')

})

client.connect(options.port, options.host, () => {

console.log('client connected');

time = process.hrtime()

client.write('hello')

})

**Листинг 4**

tcp-сервер на языке node js

'use strict'

const net = require('net');

const options = {

maxContentLength: 32768 \* 2048 + 5

}

const client = new net.Socket();

let contentLength = 0

const server = net.createServer((client) => {

console.log('client connected')

client.on('close', () => {

console.log(`total server: ${contentLength}`)

console.log('Connection was closed')

contentLength = 0

})

client.on('readable', function () {

let chunk,

buffer = null

while (null !== (chunk = client.read())) {

buffer = buffer === null

? Uint8Array.from(chunk)

: Uint8Array.concat(buffer, chunk)

}

buffer !== null && (contentLength += buffer.length)

if (contentLength < options.maxContentLength) {

return client.write(new Buffer(buffer))

}

buffer !== null && client.end(new Buffer(buffer))

})

})

server.on('error', (err) => {

server.close()

throw err;

})

server.listen(1337, () => console.log('"Waiting for a connection...'))

**Листинг 5**

http сервер на языке node js

'use strict'

const http = require('http')

const fs = require('fs')

const cluster = require('cluster')

const os = require('os')

const processorNumber = os.cpus().length

let [port, fileNumber, isLoadingIncluded, isClusterIncluded, directory] = process.argv.slice(2)

const quickSort = (array, left, right) => {

let temp;

let x = array[left + (right - left) / 2];

let i = left;

let j = right;

while (i <= j) {

while (array[i] < x) {

i++

}

while (array[j] > x) {

j--

}

if (i <= j) {

temp = array[i];

array[i] = array[j];

array[j] = temp;

i++;

j--;

}

}

if (i < right)

quickSort(array, i, right);

if (left < j)

quickSort(array, left, j);

}

const createServer = () => {

http.createServer((request, response) => {

const file = parseInt(request.url.substring(1))

let fileName = `000${file % fileNumber}`.slice(-3)

fileName = `file${fileName}.txt`

fs.readFile(`${directory}\\${fileName}`, 'ascii', (err, data) => {

if (err) {

response.writeHead(400, {'Content-Type': 'text/plain'})

response.end()

}

else {

if (isLoadingIncluded === 'true') {

const array = data.toString().split("\r\n")

const time = process.hrtime()

quickSort(array, 0, array.length - 1)

const [begin,end] = process.hrtime(time)

const interval = (begin \* 1e9 + end) / 1000000

response.writeHead(200, {'Content-Type': 'text/plain'})

response.end(`file: ${fileName}, length: ${array.length}, time: ${interval}`)

} else {

response.writeHead(200, {'Content-Type': 'text/plain'})

response.end(`file: ${fileName}; without loading`)

}

}

})

}).listen(port, '127.0.0.1')

console.log(`Server running at http://127.0.0.1:${port}/`)

}

if (isClusterIncluded === 'true') {

if (cluster.isMaster) {

for (let i = 0; i < processorNumber; i++) {

cluster.fork()

}

cluster.on('exit', (worker, code, signal) => {

console.log('worker ' + worker.process.pid + ' died')

})

}

else {

createServer()

}

} else{

createServer()

}

**Листинг 6**

http-клиент на языке C#

public class Client

{

private readonly string \_baseUrl;

private readonly int \_tasks;

private readonly ConcurrentQueue<string> \_result;

public Client(string baseUrl, int tasks)

{

\_baseUrl = baseUrl;

\_tasks = tasks;

\_result = new ConcurrentQueue<string>();

}

public void Start()

{

Console.WriteLine();

Console.WriteLine("Start sending");

var timer = new Stopwatch();

timer.Start();

var tasks = new Task[\_tasks];

for (int i = 0; i < \_tasks; ++i)

{

tasks[i] = Perform(i);

}

Task.WaitAll(tasks, -1);

timer.Stop();

//\_result.ToList().ForEach(Console.WriteLine);

Console.WriteLine();

Console.WriteLine("tasks: {0} time: {1}", \_tasks, timer.ElapsedMilliseconds);

}

private async Task Perform(int state)

{

string url = String.Format("{0}{1}", \_baseUrl, state.ToString().PadLeft(3, '0'));

var client = new HttpClient();

var timer = new Stopwatch();

timer.Start();

var stringResult = await client.GetStringAsync(url);

timer.Stop();

\_result.Enqueue(string.Format("{0,4}\t{1,5}\t{2}", url, timer.ElapsedMilliseconds, stringResult));

}

}

static class Program

{

public static void Main(string[] args)

{

var address = args[0];

for (var i = 100; i <= 500; i+=100)

{

var client = new Client(address, i);

client.Start();

}

Console.ReadLine();

}

}

**Листинг 7**

http-сервер на языке C#

public class Server

{

private readonly HttpListener \_listener = new HttpListener();

private readonly ASCIIEncoding \_encoding = new ASCIIEncoding();

private readonly string \_directory;

private readonly int \_maxFileNumber;

private readonly bool \_isLoadingIncluded;

public Server(string address, string directory, int maxFileNumber, bool isLoadingIncluded)

{

\_directory = directory;

\_maxFileNumber = maxFileNumber;

\_isLoadingIncluded = isLoadingIncluded;

\_listener.Prefixes.Add(address);

\_listener.Start();

}

public async Task Start()

{

while (true)

{

var context = await \_listener.GetContextAsync();

ProcessRequest(context);

}

}

private string GetFileName(string url)

{

var file = string.IsNullOrEmpty(url) ? 1 : int.Parse(url) % \_maxFileNumber;

return string.Format("file{0}.txt", file.ToString().PadLeft(3, '0'));

}

private async void ProcessRequest(HttpListenerContext context)

{

try

{

var filename = GetFileName(context.Request.Url.PathAndQuery.Substring(1));

string receivedData;

var path = Path.Combine(\_directory, filename);

using (var reader = new StreamReader(path))

{

receivedData = await reader.ReadToEndAsync();

}

byte[] response;

if (\_isLoadingIncluded)

{

var tuple = await SortAsync(receivedData);

response =

\_encoding.GetBytes(string.Format("file: {0}, length: {1}, time: {2}", filename,

tuple.Item1.Length, tuple.Item2));

}

else

{

response = \_encoding.GetBytes(string.Format("file: {0}; without loading", filename));

}

await context.Response.OutputStream.WriteAsync(response, 0, response.Length);

context.Response.StatusCode = (int)HttpStatusCode.OK;

}

catch (Exception e)

{

context.Response.StatusCode = (int)HttpStatusCode.BadRequest;

Console.WriteLine(e.Message);

}

finally

{

context.Response.Close();

}

}

private async Task<Tuple<string[], long>> SortAsync(string rawData)

{

return await Task.Factory.StartNew(() =>

{

var array = rawData.Split(new[] { "\r\n" }, StringSplitOptions.RemoveEmptyEntries);

var timer = new Stopwatch();

timer.Start();

QuickSort(array,0, array.Length - 1);

timer.Stop();

return new Tuple<string[], long>(array, timer.ElapsedMilliseconds);

});

}

static void QuickSort(string[] array, int left, int right)

{

string temp;

string x = array[left + (right - left) / 2];

int i = left;

int j = right;

while (i <= j)

{

while (array[i].CompareTo(x) == -1) i++;

while (array[j].CompareTo(x) == 1) j--;

if (i <= j)

{

temp = array[i];

array[i] = array[j];

array[j] = temp;

i++;

j--;

}

}

if (i < right)

QuickSort(array, i, right);

if (left < j)

QuickSort(array, left, j);

}

}

static class Program

{

public static void Main(string[] args)

{

string address = args[0];

int maxFileNumber = int.Parse(args[1]);

bool isLoadingIncluded = bool.Parse(args[2]);

string directory = args[3];

var program = new Server(address, directory, maxFileNumber, isLoadingIncluded);

program.Start().Wait();

}

}

**Листинг 8**

tcp-сервер на языке С#

public class SocketListener

{

private readonly ManualResetEvent \_threadManager = new ManualResetEvent(false);

private int \_port;

private IPAddress \_ipAdress;

public SocketListener(int port, IPAddress ipAdress)

{

\_port = port;

\_ipAdress = ipAdress;

}

public void StartListening()

{

var ipEndPoint = new IPEndPoint(\_ipAdress, \_port);

var socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp)

{

ReceiveBufferSize = StateObject.BufferSize,

SendBufferSize = StateObject.BufferSize

};

try

{

socket.Bind(ipEndPoint);

socket.Listen(100);

while (true)

{

\_threadManager.Reset();

Console.WriteLine("Waiting for a connection...");

socket.BeginAccept(AcceptCallback, socket);

\_threadManager.WaitOne();

}

}

catch (Exception e)

{

Console.WriteLine(e.ToString());

}

Console.WriteLine("\nPress ENTER to continue...");

Console.Read();

}

private void AcceptCallback(IAsyncResult ar)

{

\_threadManager.Set();

var socket = ((Socket)ar.AsyncState).EndAccept(ar);

Console.WriteLine("Client connected");

var stateObject = new StateObject { WorkSocket = socket };

socket.BeginReceive(stateObject.Buffer, 0, StateObject.BufferSize, SocketFlags.None, ReceiveCallback, stateObject);

}

private void ReceiveCallback(IAsyncResult ar)

{

var stateObject = (StateObject)ar.AsyncState;

var socket = stateObject.WorkSocket;

int bytesRead = socket.EndReceive(ar);

if (bytesRead > 0)

{

stateObject.ByteReceived += bytesRead;

socket.BeginSend(stateObject.Buffer.ToArray(), 0, bytesRead, 0, SendCallback, stateObject);

}

}

private void SendCallback(IAsyncResult ar)

{

try

{

var stateObject = (StateObject)ar.AsyncState;

var sentBytes = stateObject.WorkSocket.EndSend(ar);

stateObject.ByteSent += sentBytes;

if (stateObject.ByteSent == StateObject.MaxContentlength)

{

Console.WriteLine("Connection was closed");

if (stateObject.ByteReceived != stateObject.ByteSent)

{

throw new Exception();

}

Console.WriteLine("total : {0}",stateObject.ByteReceived);

stateObject.WorkSocket.Shutdown(SocketShutdown.Both);

stateObject.WorkSocket.Close();

}

else

{

stateObject.WorkSocket.BeginReceive(stateObject.Buffer, 0, StateObject.BufferSize, 0,

ReceiveCallback, stateObject);

}

}

catch (Exception e)

{

Console.WriteLine(e.ToString());

}

}

}

**Листинг 9**

udp-сервер на языке C#

public class UdpSocketListener

{

private readonly ManualResetEvent \_threadManager = new ManualResetEvent(false);

private EndPoint \_receiveEndPoint;

private EndPoint \_sendEndPoint;

public UdpSocketListener(int portForSending, int portForReceiving, IPAddress ipAdress)

{

\_sendEndPoint = new IPEndPoint(ipAdress, portForSending);

\_receiveEndPoint = new IPEndPoint(ipAdress, portForReceiving);

}

public void StartListening()

{

var socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp)

{

ReceiveBufferSize = StateObject.BufferSize,

SendBufferSize = StateObject.BufferSize

};

try

{

socket.Bind(\_receiveEndPoint);

while (true)

{

\_threadManager.Reset();

Console.WriteLine("Waiting for a connection...");

var stateObject = new StateObject { WorkSocket = socket };

socket.BeginReceiveFrom(stateObject.Buffer, 0, StateObject.BufferSize, SocketFlags.None,

ref \_receiveEndPoint, ReceiveCallback, stateObject);

\_threadManager.WaitOne();

}

}

catch (Exception e)

{

Console.WriteLine(e.ToString());

}

Console.WriteLine("\nPress ENTER to continue...");

Console.Read();

}

private void ReceiveCallback(IAsyncResult ar)

{

var stateObject = (StateObject)ar.AsyncState;

var socket = stateObject.WorkSocket;

int bytesRead = socket.EndReceiveFrom(ar, ref \_receiveEndPoint);

if (bytesRead > 0)

{

stateObject.ByteReceived += bytesRead;

socket.BeginSendTo(stateObject.Buffer.ToArray(), 0, bytesRead, 0, \_sendEndPoint, SendCallback, stateObject);

}

}

private void SendCallback(IAsyncResult ar)

{

try

{

var stateObject = (StateObject)ar.AsyncState;

var sentBytes = stateObject.WorkSocket.EndSendTo(ar);

stateObject.ByteSent += sentBytes;

if (stateObject.ByteSent == StateObject.MaxContentlength)

{

Console.WriteLine("Connection was closed");

if (stateObject.ByteReceived != stateObject.ByteSent)

{

throw new Exception();

}

Console.WriteLine("total : {0}", stateObject.ByteReceived);

stateObject.WorkSocket.Shutdown(SocketShutdown.Both);

stateObject.WorkSocket.Close();

}

else

{

stateObject.WorkSocket.BeginReceiveFrom(stateObject.Buffer, 0, StateObject.BufferSize, 0, ref \_receiveEndPoint,

ReceiveCallback, stateObject);

}

}

catch (Exception e)

{

Console.WriteLine(e.ToString());

}

}

}

public class StateObject

{

public Socket WorkSocket;

public readonly byte[] Buffer = new byte[BufferSize];

public const int BufferSize = 32768;

public const int MaxContentlength = 2048 \* 32768 + 5;

public long ByteReceived = 0;

public long ByteSent = 0;

}

**Листинг 10**

Класс создания текстовых файлов для нагрузки сервера

public class DataCreator

{

public void CreateFiles(int fileNumber, string directory)

{

for (int i = 0; i < fileNumber; ++i)

{

byte[] data = Generate();

var fileName = string.Format("file{0}.txt", i.ToString().PadLeft(3, '0'));

var path = Path.Combine(directory, fileName);

using (var stream = File.Open(path, FileMode.OpenOrCreate))

{

stream.Write(data, 0, data.Length);

}

}

}

private byte[] Generate()

{

var encoding = new ASCIIEncoding();

var random = new Random((int)(DateTime.UtcNow.Ticks % Int32.MaxValue) + 1);

var data = new StringBuilder();

for (long i = 0; i < 30000; ++i)

{

data.AppendLine(random.NextDouble().ToString("F7"));

}

data.Append(random.NextDouble().ToString("F7"));

return encoding.GetBytes(data.ToString());

}

}