Публикация на тему

Создание http- и tcp-серверов в Node.js

В публикации рассматриваются вопросы создания http- и tcp-серверов на Node.js

Автор
Михалькевич Александр Викторович

Публикация

Наименование Создание http- и tcp-серверов в Node.js

Автор А.В.Михалькевич

Специальность В публикации рассматриваются вопросы создания http- и tcp-серверов на Node.js,

Анотация

Anotation in English

Ключевые слова

Количество символов 8460

Содержание

Введение

Основная часть ядра Node предназначена для создания служб прослушивания. Существуют node-модули для создания HTTP-сервера, TCP-сервера, TLS-сервера, модуль UDP-сокета, или сокета дейтаграмм.

1 HTTP-сервер

Большинство стандартных web-приложений запускаются через http-протокол (HyperText Transfer Protocol – протокол передачи гипертекста).

Можно сказать, что HTTP-протокол является частным случаем протокола TCP. Так и в ядре Node.js мудуль для создания протокола HTTP (который так и называется http), наследует функциональность модуля Net (модуль протокола TCP).

Модуль HTTP представляет базовую HTTP-функциональность, обеспечивающую приложению сетевой доступ к запросам и ответам. Рассмотрим пример создания HTTP-сервера:

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

http.createServer(function(req, res){

 res.writeHead(200, {'content-type': 'text/plain'});

 res.end('Hello world!');

}).listen(8128);
console.log('Server running on 8128');
```

Набираем в браузере http://127.0.0.1:8128 и увидим на экране Hello world!

Следует обратить внимание на важную деталь: если мы запустим еще один процесс, то консоль выдаст ошибку. Система не может слушать один и тот же пор дважды.

Для того чтобы еще раз запустить прослушивание того же порта, необходимо закрыть предыдущее прослушивание.

С помощью функции createServer и безымянной функции обратного вызова создается новый сервер. Входящие параметры функции обратного вызова: req (серверный запрос или поток чтения – это объект http.serverRequest) и res (серверный ответ или поток записи – это объект http.serverResponse).

У объекта http.serverResponse имеются следующие методы:

res.writeHead(), который отправляет заголовок ответа с кодом статуса ответа.
res.end(), который подает сигнал о завершении переадчи данных и тело ответа для вывода на экран.
res.write(), который выводит данные на экран без сигнала о завершении переадчи данных.

Метод http.Server.listen прослушивает входящие подключения к заданному порту. Метод listen является асинхронным, т.е. не блокирует выполнение программы в ожидании подключения. Поэтому, функция console.log() листинга может выполниться раньше подключения.

Кроме потока чтения и записи, HTTP поддерживает кодировку фрагментированной передачи. Этот тип кодировки применяется для обработки больших объемов данных. При этом запись данных может начаться еще до получения оставшейся части запрошенных данных.

Модули Net и HTTP могут также подключаться к UNIX-сокету, а не к конкретному сетевому порту, что позволяет поддерживать взаимодействие между процессами в пределах одной и той же системы.

2 TCP-сервер

Протокол TCP (Transmission Control Protocol – протокол управления передачей) является базовым для многих интернет-приложений. По сути, протокол HTTP является частным случаем протокола TCP.

Для создания TCP-сервера и TCP-клиента, имеется встроенный модуль Net.

Мы можем создать сервер, передавая функцию обратного вызова с единственным аргументом функции – экземпляром сокета, прослушивающего два события: получение данных и закрытие соединения клиентом. Создадим в отдельном файле (например, server.js) сервер с помощью следующего листинга.

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

var server = net.createServer(function(conn){

  console.log('connected');

  conn.on('data', function(data){

      console.log( data+ ' от ' + conn.remoteAddress + ' ' +
conn.remotePort);

      conn.write(data + ' - никому.');

  });

  conn.on('close', function(){

      console.log('client closed connection');

  })
}).listen(8125);
```

Посредством метода on назначаются два прослушивателя событий. Первым параметром метод принимает имя события, вторым – функцию прослушиватель.

Создание TCP-клиента. Для этого создадим отдельный файл (например, client.js), и в нем напишем следующее:

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

var client = new net.Socket();

client.setEncoding('utf8');

client.connect(8125, 'localhost', function(){

  console.log('connected to Server');
```

```
  client.write('Кому нужен браузер?');

});

process.stdin.resume(); // подготовка к вводу данных с консоли

process.stdin.on('data', function(data){

    client.write(data);

});

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

    console.log(data);

});

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

    console.log('connection is closed');
})
```

Итак, у нас готовы два файла, один из которых является клиентом, второй – сервером.

Клиентское приложение отправляет только что набранную строку, которую сервер выводит в консоль и отвечает клиенту, дублируя эту строку и добавляя свою.

Чтобы протестировать эти node-приложения, запустим две консоли. Первым запустим приложение сервера.

Запускаем TCP-сервер:

node server.js

Откроем новую консоль, и запустим клиента:

node client.js

Соединение между клиентом и сервером поддерживается до тех пор, пока не будет прервано с одной из сторон. В режиме REPL – это комбинация клавиш Ctrl+C. Сведения об этом выводятся на консоль.

Заключение

Список использованных источников

Приложения