Search Search

Sobre Código

Desenvolvimento de Software, palestras, tutoriais e treinamentos

- **Home**
- **Contato**

« this, self e module.exports – uma introdução rápida ao Node.JS

Node.JS – servindo conteúdo dinâmico e acessando um banco de dados »

Aug 22

Um servidor de arquivo HTTP simples – mais um exemplo em Node.js

Categories:

Node.JS

by admin

Continuando a falar um pouco de node.js, vamos ver agora um dos recursos básicos da linguagem, vamos criar um servidor HTTP, e até brincar um pouco com WebSockets também, não vamos detalhar muitos recursos avançados, isto vai ser pouco mais do que uma introdução, mas eu achei interessante saber que era possível fazer isto com node.js.

O primeiro servidor HTTP com node já criamos no post anterior, foi o exemplo "Ola mundo" HTTP, que pode ser visto novamente abaixo:
samples/02_hello.js

```
1 var http = require('http');
2 http.createServer(function (req, res) {
3   res.writeHead(200, {'Content-Type': 'text/plain'});
4   res.end('Hello Worldn');
5 }).listen(1337, "127.0.0.1");
6 console.log('Server running at http://127.0.0.1:1337/');
```

Mas este não é exatamente um exemplo útil, vamos alterar este exemplo um pouquinho para que ele possa servir arquivos estáticos:
samples/06_http.js

```
1  var url = require("url"),
2      path = require("path"),
3      fs = require("fs"),
4      http = require("http");
5
6  http.createServer(function(req,res){
7    req.addListener("end",function(){
8      file = url.parse(req.url).pathname;
9      path.exists(path.join(__dirname,file), function(exists){
10       if(exists){
11         res.writeHead(200, {'Content-Type': 'text/plain'});
12         fs.readFile(path.join(__dirname,file), function (err, data) {
13           if (err) throw err;
```

```
14          res.end(data);
15        });
16      }else{
17        res.writeHead(404, {'Content-Type': 'text/plain'});
18        res.end('File not foundn');
19      }
20    });
21  });
22 }).listen(3000);
23 console.log("Listening on http://0.0.0.0:3000")
```

Existem módulos prontos na internet para servir arquivos estáticos que resolvem o problema sem criar mais problemas, uma forma interessante é colocar um outro servidor na frente do node para fazer isto como o Nginx ou o lighthttp como se faz com Rails quase sempre, mas este é um bom exemplo por que vamos começar a olhar outros modulos e vamos começar a ver bem o conceito de callbacks muito utilizado no node.js.
No código mostrado, nas linhas 1 a 4, carrego os modulos que vamos utilizar, o modulo URL vai ser utilizado para fazer o parse da URL solicitada pelo cliente, o código das linhas 6 e 22 já são velhos conhecidos, estamos criando um servidor e ouvindo em uma porta especifica, poderiamos também ter especificado o endereço IP, mas não achei que fosse necessário.
Agora vamos ao código que mudou bastante no novo servidor HTTP que criamos:
Na linha 7, estou adicionando um listener ao evento end do objeto request do servidor, este evento sera chamado sempre que um cliente conectado enviar todos os dados necessários, fizer uma requisição, ou seja, sempre que um cliente digitar o endereço no browser ou fizer um post para este servidor, e o servidor terminar de receber os dados do post.
Na linha 8, estamos utilizando o modulo request para fazer o parse da URL requisitada pelo cliente, e estamos lendo apenas a propriedade pathname, ou seja o arquivo solicitado, sem parâmetros e coisas do gênero.
Na linha 9, estamos utilizando o modulo path para várias coisas, e também a variável implicita __dirname que aponta para o diretório do arquivo.js atual, mas voltando ao modulo path, os métodos utilizados foram join para montar um nome de arquivo e exists para verificar se o arquivo existe.
Perceba que o exists também recebe uma função callback que ele chama assim que descobre se o arquivo existe ou não, como regra geral, todo e qualquer método que precise de algum tipo de operação de IO no node.js é executado de forma assincrona, liberando assim a thread do node.js para tratar de outra tarefa que não esteja bloqueada no momento.
Depois na linha 12, o modulo fs é utilizado para ler o conteúdo do arquivo, o que também utiliza um método de callback, que é chamado quando algum erro acontecer ou quando os dados do arquivo estiverem disponíveis.
Aqui é possível perceber um outro padrão do node.js, em todos os métodos que precisam de IO, e que podem gerar algum erro, o primeiro parâmetro do callback é o erro, caso existe, ou null, caso não tenha ocorrido erro.
Depois na linha 14 os dados do arquivo são enviados.
Outra coisa interessante é perceber que estamos respeitando o protocolo HTTP e retornando status 200 caso sucesso e 404 caso não encontremos o arquivo, nas linhas 11 e 17 respectivamente.

Mas esta solução como servidor de arquivos estáticos tem um problema grave, não estamos limitando o diretório abaixo do qual os arquivos podem ser lidos, por exemplo, se alguem acessar:

http://localhost:3000/../nome_do_arquivo.secret

E este arquivo existir, o servidor ira retorna-lo sem qualquer tipo de restrição, ou seja, não tentem utilizar este código em produção, ele é só um exemplo.

Então agora que já superamos esta fase, vamos utilizar um modulo pronto para isto, para limpar um pouco o código, e vamos nos concentrar nos próximos exemplos.
Instalei o modulo “node-static” com o comando “npm install node-static”, lembre de executar este comando no diretório onde ficam os arquivos .js da aplicação de exemplo, no meu caso foi no diretório samples.
Com isto agora podemos refatorar o servidor de arquivos estáticos para ficar parecido com isto:
samples/07_http.js

```
1  var nodeStatic = require('node-static'),
2      http = require('http');
3
4  var file = new nodeStatic.Server(__dirname);
5
6  http.createServer(function (request, response) {
7      request.addListener('end', function () {
8          file.serve(request, response, function (err, result) {
9              if (err) {
10                 sys.error("Error serving " + request.url + " - " + err.message);
11                 response.writeHead(err.status, err.headers);
12                 response.end();
13             }
14         });
15     });
16 }).listen(3000);
```

Este exemplo tem muito menos requires, e o código esta bem mais simples, nas linhas 1 e 2 carregamos o modulo http e o node-static que acabamos de instalar, na linha 4 criamos o servidor do node-static, ele recebe o nome do diretório de onde deve servir os arquivos estáticos como parâmetro, se nenhum parâmetro for passado o diretório atual é utilizado, ou seja, passar __dirname como fizemos é redundante, mas acho que fica mais legivel.
As linhas 6, 7 e 16 já são conhecidas, a diferença esta na linha 8 onde utilizamos o node-static para responder a requisição recebida, e no callback passado, caso um erro tenha sido gerado servindo a página podemos responder com conteúdo dinâmico, e é exatamente isto que faremos nos próximos

exemplos, mas por enquanto vamos apenas enviar o código do erro para o cliente.

Antes de começar a servir conteúdo dinâmico, vamos reorganizar as coisas um pouco, separar o exemplo em modulos e organizar melhor o código. Vamos criar um modulo que vai encapsular o servidor, e o arquivo principal vai cuidar de configuração e inicialização deste modulo, vamos começar criando o arquivo abaixo:
samples/08_server.js

```
1  var nodeStatic = require('node-static'),
2      http = require('http'),
3      sys = require('sys');
4
5  function MagicServer(config){
6      var self = this;
7      self.port = config.port ? config.port : 1337;
8      self.address = config.address ? config.address : "0.0.0.0";
9      self.public_web = config.public_web ? config.public_web : __dirname;
10     self.fileServer = new nodeStatic.Server(self.public_web);
11     self.initialize();
12 }
13
14 MagicServer.prototype.initialize = function(){
15     var self = this;
16     self.httpServer = http.createServer(function (request, response) {
17         request.addListener('end', function () {
18             self.fileServer.serve(request, response, function (err, result) {
19                 if (err) {
20                     sys.error("Error serving " + request.url + " - " + err.message);
21                     response.writeHead(err.status, err.headers);
22                     response.end();
23                 }
24             });
25         });
26     });
27     self.httpServer.listen(self.port, self.address);
28 };
29 MagicServer.prototype.stop = function(){
30     var self = this;
31     self.httpServer.stop();
32 }
33
34
35 exports.Magic = MagicServer;
```

O código é o mesmo utilizado no exemplo anterior, apenas organizado dentro de uma classe, e na linha 31 foi adicionada a chamada do método stop para parar o servidor depois do mesmo inicializado.
Entre as linhas 14 e 28 fica mais claro o por que da declaração da variável “self” no início de cada método, mais especificamente na linha 18 não seria possível acessar o servidor HTTP pois, naquele ponto, “this” aponta para um objeto request e não mais para o MagicServer como seria de se esperar, as a técnica de atribuit this a uma variável resolve este problema se forma fácil e prática.
Percebam que utilizei o operador ternário para a leitura das propriedades, normalmente não gosto de utiliza-lo, mas na minha opinião leitura de configurações com um valor default é uma das poucas ocasiões onde isto é aceitável.
A utilização do modulo criado é simples, basta criar este arquivo:
samples/09_server.js

```
1 var Server = require('./08_server');
2 var server = new Server.Magic({
3 });
```

Como parâmetro do construtor é possível alterar a porta e o diretório como o código do construtor mostra.
Agora falta adicionar suporte a conteúdo dinâmico, acho que vamos criar um framework MVC para isto, mas este vai ser o assunto para um próximo post sobre Node.js.

Se você esta gostando destes posts básicos sobre node.js deixe algum comentário, os comentários incentivam a escrita de mais posts.
E se ficou alguma dúvida pode perguntar, tentarei receber o mais rápido possível.

Tags: http, module, nodejs, server

Profile

Sign in with Twitter Sign in with Facebook
or

Name

Email Not published

Website

Message: *

Post It

- 5 Replies
- 4 Comments
- 0 Tweets
- 0 Facebook
- 1 Pingback

Last reply was September 20, 2011

1.  *Nando Vieira*
   View August 22, 2011

   Fala Urubatan!

   Acho desnecessário atribuir o this para o self em todos os casos, mesmo quando o escopo não muda, como é o caso do MagicServer e MagicServer#stop.

   Reply

   -  *Urubatan* replied:
     View August 22, 2011

     Concordo que não seja necessário em todos os casos, acho até desperdício de código, acho que ficou assim por causa do template que eu criei e acabei usando em todos os casos 😀

     Reply

     -  *Nando Vieira* replied:
       View August 24, 2011

       Hehehehee… copy ‘n’ paste fail! 😀

       Reply
2. *Um chat em node.js com websockets – o servidor » Blog do Urubatan*
   View September 12, 2011

   [...] o cliente, em parte graças aos modulos node-static e websocket-server que foram utilizados, já falamos antes sobre o node-static, e vou falar um pouquinho sobre o outro neste post, mas deixei os links caso queiram mais [...]

   Reply
3.  *cas tactics*
   View September 20, 2011

   esse seu post sobre o node.js é bem completo, estou começando agora e me está ajudando, vamos ver se não me encrenco no caminho!!! Valeu a ajuda.

   Abraço

   Reply

Search Search

Powered by Webbynode

Recent Posts

- Dica de SQL: Nunca, em hipótese alguma, use parallel via trigger de logon no oracle
- Um padrão de código para qualquer linguagem OO, sugestão e como melhorar?
- Domain Driven Design – Um rascunho de artigo
- Fui entrevistado pelo Ricardo da Revista Internet
- Slides da palestra que fiz no RS On Rails

Recent Comments

- Paulo Ricardo Araújo on Dica de SQL: Nunca, em hipótese alguma, use parallel via trigger de logon no oracle
- Alexandre Rodrigues on Dica de SQL: Nunca, em hipótese alguma, use parallel via trigger de logon no oracle
- Luciano De Mello Mattos on Um padrão de código para qualquer linguagem OO, sugestão e como melhorar?
- p_balduino on Um padrão de código para qualquer linguagem OO, sugestão e como melhorar?
- Urubatan on Um padrão de código para qualquer linguagem OO, sugestão e como melhorar?

Archives

- April 2013
- March 2013
- January 2013
- September 2012
- August 2012
- July 2012
- May 2012
- February 2012
- January 2012
- December 2011
- November 2011
- October 2011
- September 2011
- August 2011
- May 2011
- March 2011
- February 2011
- January 2011
- November 2010
- September 2010
- August 2010
- July 2010
- June 2010
- May 2010
- April 2010
- February 2010
- January 2010
- December 2009
- November 2009
- October 2009
- September 2009
- August 2009
- July 2009
- June 2009
- May 2009
- April 2009
- March 2009
- February 2009
- January 2009

- December 2008
- November 2008
- October 2008
- September 2008
- July 2008
- June 2008
- May 2008
- April 2008
- March 2008
- February 2008
- January 2008
- December 2007
- November 2007
- October 2007
- September 2007
- August 2007
- July 2007
- June 2007
- May 2007
- April 2007
- March 2007
- February 2007
- January 2007
- December 2006
- November 2006
- October 2006
- September 2006
- August 2006
- July 2006
- June 2006
- May 2006
- April 2006
- March 2006
- February 2006
- January 2006
- December 2005
- November 2005
- October 2005
- September 2005
- August 2005
- July 2005
- June 2005
- May 2005
- April 2005
- March 2005
- November 2004
- August 2004
- July 2004
- June 2004

Categories

- Artigos
- Dia a Dia
- flex
- Histórico
- Java
- Mobile
- Node.JS
- rails
- Ruby
- Scala
- Trabalho
- Uncategorized

Meta

- Log in
- Entries RSS
- Comments RSS
- WordPress.org

Copyright

- Return to top