Colin J. Ihrig

Pro Node.js para Desenvolvedores

Tradução:

Angelo Giuseppe Meira Costa (angico)

EDITORA CIÊNCIA MODERNA

Do original*:*
Pro Node.js for Developers

Editor: Paulo André P. Marques
Produção Editorial: Aline Vieira Marques
Assistente Editorial: Dilene Sandes Pessanha
Capa: Equipe Ciência Moderna (baseada no original)
Tradução: Ângelo Giuseppe Meira Costa (angico)
Diagramação: Daniel Jara

FICHA CATALOGRÁFICA

IHRIG, Colin J.

Pro Node.js para Desenvolvedores

Rio de Janeiro: Editora Ciência Moderna Ltda., 2014.

1. Linguagem de programação
I — Título

ISBN: 978-85-399-0552-2 CDD 001642

Editora Ciência Moderna Ltda.
R. Alice Figueiredo, 46 – Riachuelo
Rio de Janeiro, RJ – Brasil CEP: 20.950-150
Tel: (21) 2201-6662/ Fax: (21) 2201-6896
E-MAIL: LCM@LCM.COM.BR
WWW.LCM.COM.BR **06/14**

Sobre o Autor

Colin Ihrig tem feito experimentos com o JavaScript por prazer e por dinheiro por mais de 15 anos. Atualmente, ele é engenheiro em tempo integral com o Node.js, bem como escritor e evangelista do JavaScript, nas horas vagas. Colin recebeu seu bacharelado em Engenharia e mestrado em Engenharia da Computação da Universidade de Pittsburgh, em 2005 e 2008, respectivamente. Colin pode ser alcançado através de sua página web pessoal, em http://www.cjihrig.com.

Sobre o Revisor Técnico

Andy Olsen é consultor/treinador freelance residente no Reino Unido, e tem trabalhado em sistemas distribuídos há 20 anos. Andy começou a trabalhar em C, em meados da década de 1980, mas pode muito bem ter sido em meados da década de 1880, pois parece há muito tempo. Andy migrou para o C++, para o Java, e para o .NET conforme os tempos e as modas mudavam, e é atualmente mantido ocupado (demais?) em sistemas baseados na web, tanto do lado cliente quanto do lado servidor. Andy mora no litoral de Swansea e curte correr, cafeterias e observar os cisnes.

Agradecimentos

Eu gostaria de agradecer a todos os que ajudaram a tornar este livro possível. Agradecimentos especiais a Mark Powers e Ewan Buckingham, da equipe editorial da Apress. Também gostaria de agradecer ao revisor técnico, Andy Olsen, por seu valioso feedback. É claro que muitos agradecimentos vão para meus amigos e minha família.

Introdução

Desde sua criação, em 2009, o Node.js evoluiu até a poderosa e crescentemente popular estrutura de desenvolvimento assíncrono, usada para a criação de aplicativos JavaScript altamente escalonáveis. Empresas respeitadas como a Dow Jones, a LinkedIn e a Walmart estão entre as muitas organizações que têm visto o potencial do Node e o tem adotado em seus negócios.

O Pro Node.js para Desenvolvedores oferece um guia abrangente para essa excitante e jovem tecnologia. Você vai ser apresentado ao Node num nível alto, antes de mergulhar profundamente nos conceitos chaves e APIs que são o lastro de sua operação. Montando em cima de suas habilidades de JavaScript existentes, você vai aprender a usar o Node.js para construir aplicativos baseados tanto em web quanto em rede, para lidar com várias fontes de dados, capturar e gerar eventos, gerar e controlar processos filhos e muito mais.

Depois que tiver dominado essas habilidades, você vai receber conhecimentos mais avançados de engenharia de software, que vão dar ao seu código um nível profissional. Você vai aprender a criar facilmente módulos de código reutilizáveis, a depurar e testar seus aplicativos rápida e eficientemente e a escalonar seu código desde um único segmento até a nuvem, conforme aumente a demanda pelo seu aplicativo.

Sumário

Capítulo 1

Capítulo 2

Capítulo 3

Capítulo 4

Capítulo 5

Capítulo 6

Capítulo 7

Capítulo 8

Capítulo 9

Capítulo 10

Capítulo 11

Capítulo 12

Capítulo 13

Capítulo 14

Capítulo 15

Capítulo 16

Apêndice A

CAPÍTULO 1

Começando

O JavaScript foi chamado, inicialmente, de Mocha, quando foi desenvolvido na Netscape, em 1995, por Brendan Eich. Em setembro de 1995, liberações beta do Netscape Navigator 2.0 saíram com o Mocha, que foi rebatizado como LiveScript. Por volta de dezembro de 1995, o LiveScript, depois de ser novamente rebatizado, tornou-se o JavaScript, seu nome atual. Ainda por esse tempo, a Netscape estava trabalhando com a Sun, a empresa responsável pela criação da linguagem de programação Java. A escolha do nome JavaScript causou muita especulação. Muitas pessoas pensaram que a Netscape estava tentando pegar carona no nome Java, uma palavra quente, na época. Infelizmente, a escolha do nome causou muita confusão, já que muitos supuseram que as duas linguagens estavam de alguma forma relacionadas. Na realidade, elas tinham muito pouco em comum.

A despeito da confusão, o JavaScript se tornou uma linguagem de script do lado cliente de muito sucesso. Em resposta ao sucesso do JavaScript, a Microsoft criou sua própria implementação chamada JScript e a liberou com o Internet Explorer 3.0 em agosto de 1996. Em novembro de 1996, a Netscape submeteu o JavaScript para padronização pela Ecma International, uma organização internacional de padrões. Em junho de 1997, o JavaScript se tornou o padrão ECMA-262.

Ao longo dos anos, o JavaScript permaneceu o verdadeiro padrão para desenvolvimento do lado cliente. No entanto, no espaço do servidor, a história foi completamente diferente. Em sua maior parte, o âmbito do servidor pertencia a linguagens como o PHP e o Java. Uma série de projetos implementou o JavaScript como linguagem de servidor, mas nenhum deles teve sucesso, em particular. Dois obstáculos principais impediram a ampla adoção do JavaScript no servidor. O primeiro foi a reputação do JavaScript que era muito visto como linguagem de brincadeira, conveniente apenas para amadores. O segundo foi o precário desempenho do JavaScript, em comparação com o de algumas outras linguagens.

Contudo, o JavaScript tinha um grande trunfo em seu favor. A Web estava passando por um crescimento sem precedentes e a guerra dos navegadores estava se acirrando. Como única linguagem suportada por todos os principais navegadores, os engenhos de JavaScript começaram a receber a atenção do Google, da Apple e de outras empresas. Toda aquela atenção levou a melhoramentos imensos no desempenho do JavaScript . De repente, o JavaScript não era mais o retardatário.

A comunidade de desenvolvimento percebeu o recente poder do JavaScript e começou a criar aplicações interessantes. Em 2009, Ryan Dahl criou o Node.js, uma estrutura primariamente usada para criar servidores altamente escalonáveis para aplicações web. O Node.js, ou simplesmente Node, é escrito em C++ e em JavaScript. Para impulsionar o Node, Dahl conectou a força do motor V8 de JavaScript do Google (V8 é o engenho dentro do Google Chrome, o navegador mais popular existente). Usando o V8, os desenvolvedores podem escrever aplicativos completamente maduros – aplicativos que normalmente seriam escritos numa linguagem como o C ou o Java. Assim, com a invenção do Node, o JavaScript enfim se tornou uma linguagem de confiança do lado servidor.

O Modelo de Execução do Node

Além da velocidade, o Node apresenta um modelo de execução não convencional. Para entender como o Node é diferente, devemos compará-lo com o Apache, o popular servidor web da pilha de software Linux, Apache, MySQL e PHP (LAMP). Primeiro, o Apache processa apenas requisições HTTP, deixando que a lógica do aplicativo seja implementada numa linguagem tal como o PHP ou o Java. O Node remove uma camada da complexidade, combinando a lógica do servidor e do aplicativo num só lugar. Alguns desenvolvedores criticaram esse modelo, por eliminar a tradicional separação de foco empregada na pilha LAMP. Porém, essa abordagem também enseja ao Node uma flexibilidade sem precedentes como servidor.

O Node também difere de muitos outros servidores no uso de concorrência. Um servidor como o Apache mantém um grupo de segmentos para tratamento de conexões de clientes. Essa abordagem carece de escalabilidade, porque segmentos são bastante intensivos no uso de recursos. Além disso, um servidor ocupado, rapidamente consome todos os segmentos disponíveis; como resultado, mais segmentos, que são dispendiosos para se criar e eliminar, são iniciados. O Node, por outro lado, é executado num único segmento. Embora isso possa parecer uma má ideia, na prática, funciona bem, por causa da maneira como a maioria dos aplicativos de servidor funciona. Normalmente, um servidor recebe a requisição de um cliente, e depois realiza alguma operação de E/S de alta latência, como a leitura de um arquivo ou uma consulta à base de dados. Durante esse período, o servidor fica bloqueado, esperando que a operação de E/S seja completada. Em vez de ficar desocupado, o servidor poderia estar tratando mais requisições ou fazendo outra tarefa útil.

Em servidores tradicionais, é aceitável que um segmento não faça nada durante o bloqueio de uma operação de E/S. Entretanto, o Node só tem um segmento, e bloqueá-lo faz com que todo o servidor pare. Para minorar este problema, o Node usa quase que exclusivamente E/S sem bloqueio. Por exemplo, se o Node precisar realizar uma consulta a uma base de dados, ele simplesmente emite a consulta e passa a processar alguma outra coisa. Quando a consulta finalmente retorna, ela dispara uma função de rechamada assíncrona que é responsável por processar o resultado dessa consulta. Um exemplo desse processo em pseudocódigo é mostrado na listagem 1-1.

Listagem 1-1. Exemplo em pseudocódigo de consulta não bloqueadora a uma base de dados

```
var sql = "SELECT * FROM table";

database.query(sql, function(results) {
    // processa o resultado
});
// faz alguma outra coisa, em vez de esperar
```

O modelo de execução assíncrono e não bloqueador do Node oferece soluções de servidor extremamente escalonáveis, com o mínimo de sobrecarga. Muitas empresas importantes, incluindo a Microsoft, o LinkedIn, o Yahoo! e a gigante de vendas Walmart, tomaram conhecimento do Node e começaram a implementar projetos com ele. Por exemplo, o LinkedIn migrou toda sua pilha móvel para o Node e "passou de 15 servidores rodando com 15 instâncias (servidores virtuais) em cada máquina física, para apenas quatro instâncias que podem tratar o dobro do tráfego". O Node também recebeu significativo reconhecimento da mídia, tal como a conquista do Prêmio de Tecnologia do Ano 2012, da InfoWorld.

Instalando o Node

O primeiro passo para começar com o Node é a instalação. Esta seção vai ajudar você a aprontar o Node em sua máquina Ubuntu, OS X ou Windows. A maneira mais simples de instalar o Node é através do botão Install (*instalar*) na página base do Node, `http://nodejs.org`, mostrada na figura 1-1. Isso vai baixar os binários ou o instalador apropriado para seu sistema operacional.

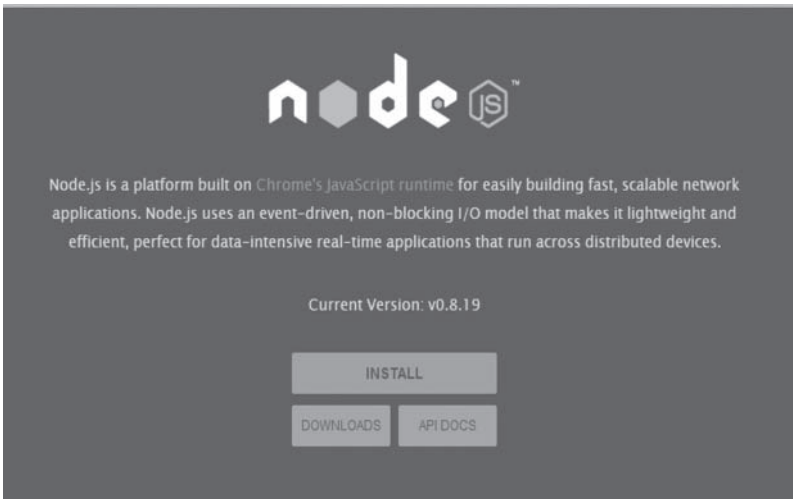

Figura 1-1. *Instalando o Node a partir da página base do projeto*

Você também pode navegar pelos binários, instaladores e pelo código fonte de todas as plataformas em `http://nodejs.org/download`. Usuários do Windows muito provavelmente vão querer baixar o Windows Installer (arquivo `.msi`), enquanto os usuários do Mac devem optar pelo Mac OS X Installer (arquivo `.pkg`). Usuários do Linux e do SunOS podem baixar binários, mas provavelmente é mais simples usar um gerenciador de pacotes.

Instalando Através de Gerenciadores de Pacotes

Para instruções sobre a instalação do Node através do gerenciador de pacotes do seu sistema operacional, vá para a página https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager. Essa página contém instruções para o Windows, o OS X e o Linux. Mais uma vez, os usuários do Windows e do Mac devem usar os instaladores discutidos anteriormente. No que diz respeito ao Linux, há instruções disponíveis para Gentoo, Debian, Linux Mint, Ubuntu, openSUSE, SLE, Red Hat, Fedora, Arch Linux, FreeBSD e OpenBSD.

Usuários do Ubuntu podem instalar o Node e todo o software necessário usando os comandos do Advanced Packaging Tool (APT) mostrados na listagem 1-2. Esses passos também instalam o `npm`, o software de gerenciamento de pacotes do Node (visto no capítulo 2).

Listagem 1-2. Instalando o Node com o gerenciador de pacotes do Ubuntu.

```
$ sudo apt-get install python-software-properties python g++ make
$ sudo add-apt-repository ppa:chris-lea/node.js
$ sudo apt-get update
$ sudo apt-get install nodejs npm
```

Se o comando `add-apt-repository` falhar, instale o pacote `software-properties-common` usando o comando mostrado na listagem 1-3.

Listagem 1-3. Instalando o pacote `software-properties-common`

```
$ sudo apt-get install software-properties-common
```

Construindo a Partir do Fonte

Se você quiser contribuir com o núcleo C++ do Node, ou simplesmente experimentar sua funcionalidade, vai precisar compilar o código fonte do projeto. Você pode obter esse código fonte a partir da página de download ou do repositório do projeto no GitHub, `https://github.com/joyent/node`. Uma vez que o código tenha sido baixado, extraia-o do arquivo, se for o caso. Antes de construir o Node, usuários do Ubuntu precisam instalar o Python e outras ferramentas de construção; use o comando mostrado na listagem 1-4. Quando da instalação do Python, certifique-se de instalar a versão 2.7, não a mais nova, Python 3.

Listagem 1-4. Instalando pacotes de software de pré-requisito, no Ubuntu

```
$ sudo apt-get install python-software-properties python g++ make
```

Usuários do Ubuntu e do OS X podem construir o Node emitindo os comandos mostrados na listagem 1-5, a partir do diretório do código fonte. Note que o caminho completo para o diretório do código fonte não deve conter nenhum espaço.

Listagem 1-5. Instalando o Node a partir do fonte no Ubuntu e no OS X

```
./configure
make
sudo make install
```

No Windows, você precisa instalar o Visual C++ e o Python 2.7 para construir o Node. O Visual C++ pode ser baixado gratuitamente da Microsoft com o Visual Studio Express. O Python também está disponível de graça, em `www.python.org/`. Para compilar o Node, emita o comando mostrado na listagem 1-6.

Listagem 1-6. Instalando o Node a partir do fonte no Windows

```
> vcbuild.bat release
```

Passos Finais da Instalação

A despeito da rota de instalação de sua escolha, a esta altura o Node já deve estar pronto para o uso. Para verificar se tudo está corretamente configurado, abra uma nova janela de terminal e rode o executável `node` (veja a listagem 1-7). O sinalizador `-v` faz com que o Node apresente a versão instalada e depois termine. Neste exemplo, a versão 0.10.18 do Node está instalada.

Listagem 1-7. Verificando a versão do Node a partir da linha de comandos

```
$ node -v
v0.10.18
```

Você também deve verificar se o `npm` está instalado (veja a listagem 1-8).

Listagem 1-8. Verificando a versão do `npm` a partir da linha de comandos

```
$ npm -v
1.3.8
```

Uma última nota a respeito da instalação: é provável que você precise instalar o Python e um compilador C++ em sua máquina, mesmo que não tenha instalado o Node a partir do fonte. Fazer isso assegura que módulos nativos escritos em C++ podem ser compilados e executados com sua instalação do Node. No Windows, isso envolve a instalação do compilador Visual C++ da Microsoft (veja a seção anterior, "Construindo a partir do fonte"). Para qualquer outro sistema operacional, a base da construção deve incluir o compilador necessário.

O Laço Ler-Avaliar-Imprimir

O Node oferece um shell interativo, conhecido como *laço Ler-Avaliar-Imprimir*, ou REPL (da sigla em inglês, *Read-Eval-Print-Loop*). O REPL lê entrada do usuário, avalia essa entrada como código JavaScript, imprime o resultado e depois espera por mais entradas. O REPL é útil para depuração e para experimentação de pequenos fragmentos de código JavaScript. Para iniciar o REPL, inicie o Node sem nenhum argumento de linha de comando. Daí, você vê o prompt de comandos do REPL, o caractere >. A partir do prompt, comece a entrar código JavaScript arbitrário.

A listagem 1-9 mostra como iniciar o REPL e entrar código. Neste exemplo, uma variável chamada `foo` é criada com o valor string "`Olá, mundo!`". Na terceira linha o REPL imprime "`undefined`", porque a sentença de declaração da variável não retorna nenhum valor. Em seguida, a sentença `foo;` faz com que o valor de `foo` seja inspecionado. Como era de se esperar, o REPL retorna a string "`Olá, mundo!`". Por fim, o valor de `foo` é impresso no terminal usando-se a função `console.log()`. Depois de `foo` ser impresso, o REPL exibe novamente "`undefined`", porque `console.log()` não retorna nenhum valor.

Listagem 1-9. Iniciando o REPL e entrando código JavaScript

```
$ node
> var foo = "Olá, mundo!";
undefined
> foo;
'Olá, mundo!'
> console.log(foo);
Olá, mundo!
undefined
```

Você também pode inserir expressões multilinhas no REPL. Por exemplo, um laço `for` introduzido no REPL na listagem 1-10. As reticências (`...`) são usadas pelo REPL para indicar uma expressão multilinhas em andamento. Note que `...` são exibidas pelo REPL e não digitadas pelo usuário.

Listagem 1-10. Um exemplo de execução de uma expressão multilinhas no REPL

```
> for (var i = 0; i < 3; i++) {
... console.log(i);
... }
0
1
2
undefined
```

Funcionalidades do REPL

O REPL tem uma série de funcionalidades que aumenta a usabilidade, das quais a mais útil é a capacidade de se navegar pelos comandos previamente emitidos, usando-se as teclas de setas para cima e para baixo. Para finalizar qualquer comando e voltar ao prompt em branco, digite `Control+C`. Pressionar `Control+C` duas vezes a partir de uma linha em branco faz com que o REPL seja encerrado. Você pode sair do REPL a qualquer momento, pressionando `Control+D`. Você pode usar a tecla `Tab` para ver uma lista de possíveis completamentos para o comando atual. Se só houver uma opção possível, o Node a insere automaticamente. A lista inclui palavras-chave, funções e variáveis. Por exemplo, a listagem 1-11 mostra as opções de completamento quando um `t` é introduzido no prompt.

Listagem 1-11. Opções de autocompletamento mostradas pela digitação de um `t` seguido de um `Tab`

```
> t
this            throw           true            try
typeof          tls             tty             toLocaleString
toString
```

O REPL também oferece uma variável especial, _ (sublinha), que sempre contém o resultado da última expressão. A listagem 1-12 mostra vários exemplos de uso de _. Primeiro, uma matriz de strings é criada, fazendo com que _ referencie a matriz. O método `pop()` é então usado para remover o último elemento da matriz `baz`. Por fim, o comprimento de `baz` é acessado, fazendo com que _ se torne `3`.

Listagem 1-12. Exemplos de uso da variável _

```
> ["foo", "bar", "baz"]
[ 'foo', 'bar', 'baz' ]
> _.pop();
'baz'
> _.length
3
> _
3
```

Comandos do REPL

`.help`

O comando `.help` exibe todos os comandos disponíveis no REPL. A listagem 1-13 mostra a saída da execução do comando `.help`.

Listagem 1-13. Saída do comando `.help` do REPL

```
> .help
.break Sometimes you get stuck, this gets you out
```

```
.clear Alias for .break
.exit Exit the repl
.help Show repl options
.load Load JS from a file into the REPL session
.save Save all evaluated commands in this REPL session to a file
```

.exit

O comando `.exit` termina o REPL. Este comando é equivalente ao pressionamento de `Control+D`.

.break

O comando `.break`, usado para abortar uma expressão multilinhas, é útil se você cometer um erro ou simplesmente optar por não completar a expressão. A listagem 1-14 mostra um exemplo de uso do comando `.break` para terminar um laço `for` antes de completá-lo. Note que o prompt normal > é mostrado após o comando `.break`.

Listagem 1-14. Terminando uma expressão multilinhas usando o comando `.break`

```
> for (var i = 0; i < 10; i++) {
... .break
>
```

.save nome_do_arquivo

O comando `.save` salva a sessão atual do REPL no arquivo especificado em `nome_do_arquivo`. Se o arquivo não existir, ele é criado. Se ele existir, ele é sobreposto. Os comandos do REPL e as saídas não são salvos. A listagem 1-15 mostra um exemplo de uso do comando `.save`. Nesse exemplo, a sessão atual é salva no arquivo `repl-test.js`. O conteúdo resultante de `repl-test.js` é mostrado na listagem 1-16. Note que o arquivo não contém o prompt do REPL, nem saídas, nem o comando `.save`.

Listagem 1-15. Salvando a sessão atual do REPL usando o comando `.save`

```
> var foo = [1, 2, 3];
undefined
> foo.forEach(function(value) {
... console.log(value);
... });
1
2
3
undefined
```

```
> .save repl-test.js
Session saved to:repl-test.js
```

Listagem 1-16. O conteúdo de `repl-test.js` gerado pelo comando `.save`

```
var foo = [1, 2, 3];
foo.forEach(function(value) {
console.log(value);
});
```

`.load nome_do_arquivo`

O comando `.load` executa o arquivo JavaScript especificado em `nome_do_arquivo`. O arquivo é executado como se cada linha fosse digitada diretamente no REPL. A listagem 1-17 mostra a saída do carregamento do arquivo `repl-test.js` da listagem 1-16.

Listagem 1-17. O resultado da execução de `repl-test.js`, usando-se o comando `.load`

```
> .load repl-test.js
> var foo = [1, 2, 3];
undefined
> foo.forEach(function(value) {
... console.log(value);
... });
1
2
3
undefined
```

`.clear`

Similar a `.break`, `.clear` pode ser usado para encerrar expressões multilinhas. `.clear` também é usado para reiniciar o objeto de contexto do REPL. Neste momento, você não precisa entender os detalhes, mas a listagem 1-18 mostra um programa Node que incorpora um REPL. Em outras palavras, ao rodar esse programa, invoca-se, na verdade, uma instância do REPL. Além disso, você pode definir um ambiente de execução personalizado para o REPL. Nesse caso, o REPL incorporado tem uma variável definida, `foo`, que guarda a string "`Olá, REPL`". Chamar `.clear` de dentro do REPL incorporado reseta o contexto e exclui `foo`.

Listagem 1-18. Embutindo um REPL dentro de outro programa Node

```
var repl = require("repl");

repl.start({}).context.foo = "Olá, REPL";
```

Executando Programas Node

Embora o ambiente REPL seja útil, ele raramente é usado em sistemas de produção. Em vez disso, os programas são escritos como um ou mais arquivos JavaScript e depois interpretados pelo Node. O programa Node mais simples é mostrado na listagem 1-19. O exemplo simplesmente imprime a string "`Olá, mundo!`" no console.

Listagem 1-19. Código fonte para o programa Node Olá, mundo!

```
console.log(«Olá, mundo!»);
```

Copie o código da listagem 1-19 para um novo arquivo e salve-o como ola.js. Em seguida, abra uma janela de terminal e execute-o (veja a listagem 1-20). Note que o Node não exige que você especifique a extensão .js. Se o arquivo de entrada não for encontrado e nenhuma extensão for fornecida, o Node tenta adicionar as extensões .js, .json, e .node. O Node interpreta arquivos .js como código fonte JavaScript e arquivos com extensão .json como arquivos de Notação de Objeto JavaScript (JSON, na sigla em inglês). Arquivos com extensão .node são tratados como módulos complementares compilados.

Listagem 1-20. Executando um programa Node a partir da linha de comandos

```
$ node ola.js
```

■ **Nota** O JSON é um texto plano padrão para intercâmbio de dados. Este livro considera que o leitor está familiarizado com o JSON. No entanto, se você precisar de uma introdução ou de uma revisão, o JSON é abordado no apêndice A.

Resumo

Parabéns! Você deu os primeiros passos em direção ao desenvolvimento de aplicativos Node. Esse capítulo ofereceu uma introdução de alto nível do Node, e guiou você pelo processo de instalação. Você até escreveu algum código Node usando o REPL. O resto deste livro se ergue sobre esse capítulo, cobrindo os aspectos mais importantes do desenvolvimento em Node. O Node é melhor conhecido pela criação de servidores web escalonáveis. Então, é claro que essa funcionalidade é abordada. Porém, você também vai aprender muito mais, incluindo a programação para sistema de arquivos, o fluxo de dados, o escalonamento de aplicativos e o sistema de módulos do Node.