Perguntas Fáceis (15)

1. O que é Node.js?
2. Qual é a principal linguagem utilizada no Node.js?
3. Node.js é baseado em qual motor de execução JavaScript?

1-2-3 – Um server para rodar código javascript do lado do servidor, sem a necessidade de um navegador. Ele faz isso através do uso do motor v8 do google.

4. Qual comando é usado para inicializar um projeto Node.js?

npm init;

5. Qual arquivo padrão contém as informações básicas de um projeto Node.js?

package.json;

6. Como se chama o sistema de módulos do Node.js?

O sistema de módulos do Node é o CommonJS que foi usado por muito tempo, porem hoje o mais usado é o ECMAScript Modules (ESM) com suporte nativo e padrão para modulo js.

CommonJS – module.exports e o require

```
module.exports = saudacao;
// main.js const saudacao = require('./module');
console.log(saudacao('Mundo'));
```

ECMAScript Modules (ESM)

```
// module.mjs
export function saudacao(nome)
 {  return `Olá, ${nome}!`;
}
// main.mjs


import { saudacao } from './module.mjs';
console.log(saudacao('Mundo')); // Olá, Mundo!
```

Resumo das Diferenças:

Característica	CommonJS	ECMAScript Modules (ESM)
Importação	require	import
Exportação	module.exports	export
Assíncrono	Não	Sim
Suporte a padrão JS	Não	Sim

Ambos os sistemas podem ser usados no Node.js moderno, dependendo do contexto e das configurações do projeto.

7. Qual comando é usado para instalar pacotes no Node.js?

npm install;

8. Qual extensão de arquivo é usada para scripts Node.js?
A extensão é .js

9. Qual é o método usado para exibir informações no console?

console.log()

10. O Node.js é single-thread ou multi-thread?

O Node.js é single-threaded, assíncrono e não bloqueante. Isso significa que ele utiliza um único thread principal para gerenciar
a execução do código, mas delega operações de I/O, como leitura de arquivos ou requisições de rede, para a thread pool ou APIs do sistema.
As tarefas assíncronas são colocadas em uma fila de eventos (Event Queue) e processadas pelo loop de eventos (Event Loop) à medida que ficam prontas.
Enquanto isso, o Node.js continua recebendo novas solicitações e as adicionando à fila, sem bloquear a execução do restante do código."

11. Qual é o principal objetivo do npm?

O principal objetivo é ser o gerenciador de pacotes de dependencias nativo do node;

12. Qual biblioteca do Node.js é usada para criar um servidor HTTP simples?

biblioteca http é usada para criar um servidor HTTP simples.

13. Qual comando é usado para rodar um arquivo JavaScript no Node.js?

node "nomeArquivo.js"

14. Qual a diferença entre require e import no Node.js?

require é usado no CommonJS, enquanto import é usado em ES Modules (ECMAScript Modules).

15. O Node.js é adequado para construir aplicações em tempo real?

Sim, o Node.js é ideal para aplicações em tempo real devido ao seu modelo assíncrono.

Perguntas Médias (15)

1. Qual é a função do método fs.readFile no Node.js?

O método fs.readFile é usado para ler o conteúdo de um arquivo de forma assíncrona.

2. O que significa o conceito de "event loop" no Node.js?

O "event loop" é responsável por gerenciar operações assíncronas e manter a aplicação não bloqueante.

3. Como lidar com erros em operações assíncronas no Node.js?

Os erros podem ser tratados com try...catch para async/await ou utilizando o parâmetro de erro em callbacks.

4. Explique o que é um "callback" no contexto do Node.js.

No contexto do Node.js, um callback é uma função passada como argumento para outra função.

5. Qual biblioteca no Node.js é comumente usada para criar rotas em aplicações web?

A biblioteca mais usada para rotas é o Express.js.

6. O que é o módulo os no Node.js e como ele pode ser usado?

O módulo os fornece informações sobre o sistema operacional, como arquitetura e memória.

7. Como é possível criar variáveis de ambiente em um projeto Node.js?

As variáveis de ambiente podem ser configuradas usando a biblioteca dotenv ou diretamente no arquivo process.env.

8. Explique a diferença entre os métodos fs.writeFile e fs.appendFile.

fs.writeFile substitui o conteúdo do arquivo, enquanto

fs.appendFile adiciona conteúdo ao final.

9. Qual é a finalidade do método process.nextTick?

O process.nextTick insere uma função na próxima fase do ciclo de eventos antes do processamento da fila.

10. Qual é a diferença entre setTimeout e setImmediate?

setTimeout agenda uma função após um atraso, e apos o fim do ciclo do script.

enquanto setImmediate executa a função na próxima iteração do loop, passado a frente na fila de execução.

11. Como o Node.js gerencia operações de I/O?

O Node.js gerencia operações de I/O (Entrada/Saída) de maneira eficiente por meio do event loop e da biblioteca libuv,
que juntos permitem a execução não bloqueante e assíncrona dessas operações.

12. Qual comando é usado para instalar uma dependência como devDependency no Node.js?

npm i "nomedpendencia" --salve-dev
O comando npm install nomeDoPacote --save-dev instala uma dependência como devDependency.

13. Explique a estrutura de middleware em aplicações Node.js usando Express.js.

- Interceptadores - ficam entre a requisição e a rota ou na ida ou na vinda de uma requisição.
Middlewares em Express.js são funções que processam requisições antes de chegarem à rota final ou
de retornarem uma resposta.

14. Qual a diferença entre o uso de res.send e res.json no Express.js?

res.send envia respostas de qualquer tipo, enquanto res.json é usado para enviar objetos JSON.

15. O que são streams no Node.js e como eles funcionam?

Streams são interfaces que manipulam dados de maneira eficiente, dividindo-os em pequenos chunks.

No Node.js, streams são abstrações para manipular dados assíncronos e contínuos.
Eles permitem processar dados em blocos (chunks), em vez de carregar tudo na memória de uma vez,
o que os torna eficientes para manipular grandes volumes de dados, como arquivos, requisições HTTP, ou
dados de rede.

1. Tipos de Streams
Os streams no Node.js podem ser classificados em quatro tipos:

Readable Streams (Streams de leitura):
Permitem ler dados de uma fonte (como um arquivo ou uma requisição HTTP).
Exemplos: fs.createReadStream, requisições HTTP (req em um servidor).

Writable Streams (Streams de escrita):
Permitem escrever dados em um destino (como um arquivo ou socket).
Exemplos: fs.createWriteStream, resposta HTTP (res em um servidor).

Duplex Streams:
Permitem ler e escrever dados, funcionando como streams bidirecionais.
Exemplo: net.Socket.

Transform Streams:
São duplex streams com uma função de transformação, permitindo modificar os dados durante a passagem.
Exemplo: zlib.createGzip (para compressão).

Perguntas Difíceis (15)

1. Explique como o "single-threaded event loop" gerencia múltiplas conexões simultaneamente.

O event loop do Node.js gerencia múltiplas conexões ao lidar com tarefas de forma assíncrona.
Ele utiliza uma única thread para executar JavaScript e delega operações bloqueantes, como acesso a
arquivos ou bancos de dados, a bibliotecas externas.

Enquanto essas operações estão em andamento, o event loop continua processando outras conexões. Quando uma operação é concluída, o Node.js chama a função correspondente (callback), sem bloquear a execução de outras tarefas. Isso permite gerenciar milhares de conexões simultaneamente com eficiência.

2. Qual é a diferença entre programação baseada em callbacks, promises e async/await no Node.js?

No Node.js, callbacks, promises e async/await são formas de lidar com operações assíncronas. Aqui estão as diferenças entre eles:

1. Callbacks

O que são? Funções passadas como argumento para serem chamadas quando uma operação assíncrona é concluída.

Vantagens:

- Simples de implementar.
- Compatível com versões antigas do Node.js.

Desvantagens:

- Leva ao callback hell (código difícil de ler e manter).
- Propagação de erros é complexa.

Exemplo:

```
fs.readFile('arquivo.txt', (err, data) => {
if (err) return console.error(err);
console.log(data.toString());
});
```

2. Promises

O que são? Objetos que representam a eventual conclusão (ou falha) de uma operação assíncrona.

Vantagens:

- Evita callback hell com encadeamento de .then() e .catch().
- Melhor suporte para tratamento de erros.

Desvantagens:

- Encadeamento extenso ainda pode ser difícil de ler.

Exemplo:

```
fs.promises.readFile('arquivo.txt')
 .then(data => console.log(data.toString()))
 .catch(err => console.error(err));
```

3. Async/Await

O que é? Sintaxe que permite escrever código assíncrono de forma mais legível, como se fosse síncrono.

Vantagens:

- Código mais simples e fácil de ler.
- Tratamento de erros com try/catch.
- Desvantagens:

Pode dificultar paralelismo em certos casos se não usado com cuidado.

Exemplo:

```
async function lerArquivo() {
  try {
    const data = await fs.promises.readFile('arquivo.txt');
          console.log(data.toString());
        } catch (err) {
    console.error(err);
```

```
          }
        }

        Aspecto              | Callbacks                   |
Promises           | Async/Await

---------------------|-----------------------------|----------
----------|------------------
        Legibilidade         | Baixa (callback hell)       |
Moderada           | Alta
        Tratamento de erros  | Complexo                    |
Simples (.catch()) | Simples (try/catch)
        Complexidade         | Simples, mas desorganizado  |
Organizado         | Muito organizado
        Versão do Node.js    | Sempre suportado            |
Suportado desde ES6| Suportado desde ES8
```

3. Como funciona o processo de "clustering" no Node.js para aplicações multi-thread?

O clustering no Node.js permite usar múltiplos núcleos de CPU para processar conexões simultaneamente, superando a limitação do modelo single-threaded. Ele cria vários processos (workers), cada um executando uma cópia da aplicação, enquanto o processo principal (master) gerencia os workers e distribui as conexões.

Como Funciona:

- O master verifica o número de núcleos da CPU.
- Cria um worker para cada núcleo.
- O sistema operacional distribui as conexões entre os workers.

4. Explique o funcionamento interno do módulo require no Node.js.

O módulo require do Node.js é usado para importar módulos e arquivos em uma aplicação.
Ele faz parte do sistema de módulos CommonJS, que é a base do gerenciamento de
dependências no Node.js.

5. Qual é o propósito da biblioteca cluster no Node.js?

O propósito da biblioteca cluster no Node.js é permitir que uma aplicação aproveite
todos os núcleos disponíveis em um processador para melhorar o desempenho e a capacidade de
lidar com conexões simultâneas.

6. Como proteger uma aplicação Node.js contra ataques de injeção SQL?

Proteger uma aplicação Node.js contra ataques de injeção SQL envolve várias práticas recomendadas, desde o uso de ferramentas específicas até uma abordagem defensiva no desenvolvimento. Aqui estão algumas medidas que você pode tomar:

1. Use um ORM/ODM ou Query Builder

2. Sanitize e Parametrize suas Consultas

3. Valide e Limite Entradas de Usuários

4. Use Módulos de Banco de Dados Atualizados

5. Aplique Privilégios Restritos no Banco de Dados

6. Monitore e Faça Log de Consultas

Use ferramentas como Winston ou Morgan para logar e monitorar consultas SQL suspeitas ou fora do padrão.

7. Implemente Proteções Adicionais

8. Evite Dados de Entrada Diretamente no Código

A principal abordagem para prevenir ataques de injeção SQL é usar ferramentas seguras como ORMs,
parametrizar consultas, validar entradas e limitar privilégios no banco de dados.
Além disso, revisar regularmente o código e testar a aplicação contra essas vulnerabilidades é essencial.

7. Explique como funciona o mecanismo de Garbage Collection do Node.js.

O mecanismo de Garbage Collection (GC) do Node.js gerencia automaticamente a memória alocada
durante a execução do programa, liberando recursos que não são mais utilizados. Ele é baseado no V8
JavaScript Engine (usado também no Google Chrome), que implementa um algoritmo de coleta de lixo otimizado.

8. Qual é a diferença entre o Buffer e Stream no Node.js?

Resumo
- Buffer: Armazena dados binários na memória como um bloco único.
  Ideal para manipular pequenos conjuntos de dados completos.

- Stream: Processa dados em partes contínuas (chunks).

É eficiente para grandes volumes de dados, como arquivos ou conexões.

Aspecto	Buffer	Stream
Modo de Operação	Armazena tudo na memória de uma vez	Processa dados em partes contínuas
Uso de Memória	Requer toda a memória necessária	Eficiente, usa menos memória
Tamanho de Dados	Ideal para pequenos volumes	Ideal para grandes volumes
Exemplo	Arquivo pequeno carregado inteiro	Leitura de um vídeo ou API contínua

9. O que é o EventEmitter e como ele é usado?

O EventEmitter é um módulo do Node.js para criar e gerenciar eventos. Ele permite associar funções (listeners) a eventos específicos.

Métodos principais:
.on(event, listener): Associa um listener ao evento.
.emit(event, args): Dispara um evento com argumentos.
.once(event, listener): Escuta o evento apenas uma vez.

+--+

Exemplo básico:

```
const EventEmitter = require('events');
const emitter = new EventEmitter();

emitter.on('greet', (name) => {
  console.log(`Olá, ${name}!`);
});

emitter.emit('greet', 'João'); // Output: Olá, João!
```

+--+

10. Explique como configurar um servidor HTTPS no Node.js.

Um servidor HTTPS pode ser configurado no Node.js usando o módulo https com certificados SSL/TLS.

11. Como o Node.js lida com operações assíncronas que dependem de operações de bloqueio?

O Node.js lida com operações assíncronas e de bloqueio usando um modelo de I/O não bloqueante.
Ele delega tarefas de bloqueio (como leitura de arquivos) ao thread pool da libuv, permitindo que
o event loop continue processando outras tarefas. Se operações síncronas (bloqueantes) forem usadas,
como fs.readFileSync(), elas podem travar o event loop e afetar a performance. O uso de Promises e async/await
facilita o código assíncrono sem bloquear a execução.

12. Explique o que é o V8 no contexto do Node.js.

13. Como implementar autenticação JWT em uma API Node.js?
14. Qual a finalidade do módulo worker_threads no Node.js?
15. Explique a diferença entre libuv e event loop no Node.js.

Respostas - Perguntas Fáceis

1. Node.js é um ambiente de execução JavaScript para desenvolvimento de aplicações no lado do servidor.
2. A principal linguagem utilizada no Node.js é o JavaScript.
3. O Node.js é baseado no motor de execução V8, desenvolvido pelo Google.
4. O comando npm init inicializa um projeto Node.js.
5. O arquivo padrão é o package.json.
6. O sistema de módulos do Node.js é chamado de CommonJS.
7. O comando npm install é usado para instalar pacotes.
8. A extensão padrão é .js.
9. O método console.log é usado para exibir informações no console.

10. O Node.js é single-thread, mas utiliza um modelo assíncrono para gerenciar múltiplas conexões.
11. O npm (Node Package Manager) é usado para gerenciar pacotes e dependências.
12. A biblioteca http é usada para criar um servidor HTTP simples.
13. O comando node nomeDoArquivo.js é usado para executar um arquivo.
14. require é usado no CommonJS, enquanto import é usado em ES Modules (ECMAScript Modules).
15. Sim, o Node.js é ideal para aplicações em tempo real devido ao seu modelo assíncrono.

Respostas - Perguntas Médias

1. O método fs.readFile é usado para ler o conteúdo de um arquivo de forma assíncrona.
2. O "event loop" é responsável por gerenciar operações assíncronas e manter a aplicação não bloqueante.
3. Os erros podem ser tratados com try...catch para async/await ou utilizando o parâmetro de erro em callbacks.
4. Um callback é uma função passada como argumento que será executada após outra função ser concluída.
5. A biblioteca mais usada para rotas é o Express.js.
6. O módulo os fornece informações sobre o sistema operacional, como arquitetura e memória.
7. As variáveis de ambiente podem ser configuradas usando a biblioteca dotenv ou diretamente no arquivo process.env.
8. fs.writeFile substitui o conteúdo do arquivo, enquanto fs.appendFile adiciona conteúdo ao final.
9. O process.nextTick insere uma função na próxima fase do ciclo de eventos antes do processamento da fila.
10. setTimeout agenda uma função após um atraso, enquanto setImmediate executa a função na próxima iteração do loop.
11. O Node.js gerencia operações de I/O de forma assíncrona usando callbacks, Promises e o event loop.
12. O comando npm install nomeDoPacote --save-dev instala uma dependência como devDependency.
13. Middlewares em Express.js são funções que processam requisições antes de chegarem à rota final ou de retornarem uma resposta.
14. res.send envia respostas de qualquer tipo, enquanto res.json é usado para enviar objetos JSON.

15. Streams são interfaces que manipulam dados de maneira eficiente, dividindo-os em pequenos chunks.

Respostas - Perguntas Difíceis

1. O event loop gerencia múltiplas conexões simultaneamente através de callbacks e um modelo assíncrono não bloqueante.
2. Callbacks são funções passadas para serem executadas após uma operação; Promises são objetos que representam uma operação assíncrona; async/await permite um fluxo de código mais legível.
3. O clustering utiliza múltiplas instâncias do Node.js para distribuir carga entre diferentes núcleos do processador.
4. O require resolve caminhos, carrega o módulo em cache e executa o código exportado.
5. A biblioteca cluster permite criar múltiplos processos para aplicações Node.js, aproveitando vários núcleos.
6. Para proteger contra injeção SQL, usa-se query builders como Sequelize ou ORM com parametrização de consultas.
7. O Garbage Collection do Node.js gerencia a alocação e liberação de memória automaticamente.
8. O Buffer armazena dados binários diretamente na memória, enquanto Streams processam esses dados em chunks.
9. O EventEmitter é uma classe que permite criar e manipular eventos personalizados no Node.js.
10. Um servidor HTTPS pode ser configurado no Node.js usando o módulo https com certificados SSL/TLS.
11. Operações de bloqueio são minimizadas com estratégias como delegação a threads no libuv ou Promises assíncronas.
12. O V8 é o motor que compila JavaScript em código máquina e executa no Node.js.
13. A autenticação JWT pode ser implementada usando bibliotecas como jsonwebtoken para gerar e verificar tokens.
14. O módulo worker_threads permite criar threads para tarefas intensivas.
15. O libuv é uma biblioteca que implementa o event loop e fornece APIs de baixo nível, enquanto o event loop é o mecanismo que gerencia eventos.