Cainã Bortotti
@bybortotti

5 ARMADILHAS DE MEMÓRIA QUE TODO DEV NODE.JS PRECISA CONHECER

PEQUENOS DESCUIDOS PODEM TRAVAR TODO SEU SISTEMA

Cainã Bortotti
@bybortotti

Trabalhando com Node.js em sistemas de alto tráfego, percebi que erros de memória não aparecem de imediato. **Eles se acumulam, degradam performance e podem derrubar sistemas críticos.**

Ao longo de inúmeros projetos, identifiquei padrões que causam vazamentos silenciosos.

Com base nisso, essas são as cinco armadilhas mais comuns e suas consequências.

Cainã Bortotti
@bybortotti

ARMADILHA 1: VARIÁVEIS GLOBAIS NÃO INTENCIONAIS

Declarar variáveis sem `const`, `let` ou `var` as torna globais sem querer.

Problemas e consequências: Referências desnecessárias no escopo global, crescimento invisível da memória e risco de sobrescrever dados importantes.

Como evitar: Declare todas as variáveis explicitamente e use ferramentas de linting como ESLint para detectar variáveis globais acidentais.

Cainã Bortotti
@bybortotti

ARMADILHA 2: EVENT LISTENERS ACUMULANDO

Listeners em `EventEmitter` que nunca são removidos ficam ativos na memória.

Problemas e consequências: Vazamento silencioso, aumento gradual do consumo de memória, risco de travar a aplicação.

Como evitar: Sempre remova listeners que não são mais necessários com `.off()` ou `.removeListener()`. Monitore a quantidade de listeners ativos em produção.

Cainã Bortotti
@bybortotti

ARMADILHA 3: CLOSURES MATENDO REFERÊNCIAS GRANDES

Funções internas que fecham sobre objetos grandes podem impedir que o garbage collector libere memória.

Problemas e consequências: Lentidão crescente, aumento do heap e eventual travamento da aplicação em cenários de alta carga. Já vi closures em loops manter arrays gigantes vivos durante minutos, causando lentidão absurda.

Como evitar: Evite capturar objetos grandes dentro de closures e libere referências quando não forem mais necessárias.

Cainã Bortotti
@bybortotti

ARMADILHA 4: CACHES MAL CONTROLADOS

Armazenar dados em memória sem limite pode parecer eficiente, mas cresce indefinidamente.

Problemas e consequências: Memória consumida rapidamente, degradação do desempenho e risco de crash do servidor.

Como evitar: Estabeleça limites de tamanho e tempo de vida (TTL) para caches. Para grandes volumes de dados, use Redis ou Memcached.

Cainã Bortotti
@bybortotti

ARMADILHA 5: TIMES INTERVALS ESQUECIDOS

`setInterval` e `setTimeout` mantêm referências enquanto estiverem ativos

Problemas e consequências: Objetos e funções não liberados, vazamentos contínuos de memória e travamento em testes de carga. Em um projeto que acompanhei, um script de monitoramento deixou intervals ativos e gerou vazamentos que só percebemos em testes de carga.

Como evitar: Limpe timers com `clearInterval` ou `clearTimeout` quando não forem mais necessários ou quando o contexto que os usa for descartado.

Cainã Bortotti
@bybortotti

Vazamentos de memória em Node.js se acumulam silenciosamente e podem derrubar sistemas críticos.

Prestar atenção a padrões de código e revisar continuamente ajuda a prevenir problemas antes que eles apareçam. Manter a prática de monitoramento de memória e profiling é o que separa sistemas robustos de sistemas problemáticos.

Qual dessas armadilhas você já enfrentou em produção? Como resolveu? Deixe sua resposta nos comentários