Otimização do Puppeteer para Web Scraping

Este guia detalha as melhores práticas e configurações para otimizar o desempenho do Puppeteer, especialmente em cenários de web scraping.

1. Bypassing Anti-Bot Detection

Websites podem detectar e bloquear scrapers. Para evitar isso:

- **Plugins Stealth:** Use `puppeteer-extra-plugin-stealth` para mascarar impressões digitais do navegador. Instale com `npm install puppeteer-extra-plugin-stealth` e integre com `puppeteer.use(StealthPlugin())`.
- **Rotação de IPs:** Use proxies para rotacionar endereços IP e evitar bloqueios por volume de requisições.
- **Rotação de User-Agents:** Alterne os user-agents para cada requisição para evitar detecção. O plugin `puppeteer-extra-plugin-anonymize-ua` pode ajudar nisso.
- **Bloqueio de Recursos:** Use ad blockers ou bloqueie o carregamento de imagens para reduzir pixels de rastreamento e conteúdo de marketing.

2. Otimizando a Velocidade de Carregamento e Navegação da Página

Tempos de carregamento lentos impactam a velocidade do scraping. Dicas:

- **Otimizações de Carregamento:** Desabilite imagens, CSS e JavaScript não essenciais para a extração de dados. Isso reduz a quantidade de dados a serem carregados.
- **Opções `page.goto()`:** Defina um timeout e espere até que os elementos necessários estejam visíveis antes de prosseguir com a extração. Use `waitUntil: 'domcontentloaded'` para esperar o HTML inicial ou `page.waitForSelector()` para elementos específicos.
- **Intercepção de Requisições:** Intercepte e bloqueie tipos de recursos desnecessários (ex: imagens, fontes, stylesheets) usando `page.setRequestInterception(true)` e `request.abort()`.

3. Seleção e Interações Eficientes de Elementos

Interações ineficientes com o DOM podem degradar o desempenho:

- **Espera por Carregamento:** Use `timeout`, `waitForSelector` ou `waitUntil` para garantir que a página esteja completamente carregada.

- **Seletores Eficientes:** Prefira seletores como ID ou nomes de classe em vez de expressões XPath, que são mais lentas.
- **Agrupamento de Ações:** Minimize interações repetitivas agrupando ações em uma única função `evaluate()` para reduzir viagens de ida e volta entre o Node.js e o Chromium.

4. Gerenciamento de Uso de Memória e Consumo de Recursos

Gerenciar a memória é crucial para evitar vazamentos e uso excessivo de CPU:

- **Fechamento de Recursos:** Feche e descarte páginas, frames e instâncias de navegador desnecessárias. Isso libera recursos eficientemente.
- **Tratamento de Erros:** Implemente tratamento de erros e procedimentos de limpeza adequados para garantir que os recursos sejam liberados após cada operação de scraping, mesmo em caso de erro.

5. Utilizando Cache e Interceptação de Requisições

Evite requisições desnecessárias usando cache:

- **Interceptação de Requisições:** Configure a interceptação de requisições para verificar se a URL solicitada já está no cache. Se sim, sirva a resposta do cache; caso contrário, continue com a requisição original e atualize o cache.

6. Escalabilidade e Processamento Paralelo com Puppeteer

Para tarefas de scraping em larga escala, o processamento sequencial não é recomendado:

- **`puppeteer-cluster`:** Use a biblioteca `puppeteer-cluster` para criar um cluster de workers do Puppeteer, permitindo executar múltiplas tarefas simultaneamente. Instale com `npm install puppeteer-cluster`.

7. Medição e Monitoramento de Desempenho

Monitore o desempenho para identificar gargalos:

- **Chrome DevTools:** Ferramenta integrada no Chrome para monitorar atividades de rede, uso de memória e visualizar linhas do tempo de desempenho.
- **Lighthouse:** Ferramenta de código aberto para avaliar velocidade, desempenho e experiência do usuário da página.

- **APIs de Desempenho Web:** Use APIs como `Navigation Timing API` para coletar métricas de tempo de carregamento e tempo de resposta.

Métricas Chave:

1. **Tempo Total de Scraping:** Tempo total para extrair os dados.
2. **Tempo de Requisição de Rede:** Tempo para buscar dados dos sites.
3. **Tempo de Manipulação do DOM:** Tempo para processar e manipular os dados extraídos.
4. **Uso de Memória:** Monitore o consumo de memória para detectar vazamentos.

8. Opções de Lançamento do Puppeteer (`LaunchOptions`)

As opções de lançamento do Puppeteer são cruciais para otimizar o desempenho e o consumo de recursos. Algumas das opções mais importantes incluem:

- **`headless`** : Define se o navegador será executado em modo headless (sem interface gráfica). O padrão é `true` , o que é ideal para desempenho em servidores. Para depuração, pode ser definido como `false` .
- **`args`** : Permite passar argumentos de linha de comando adicionais para a instância do navegador. Argumentos úteis para otimização incluem:
  - `--no-sandbox` : Necessário ao executar Puppeteer como root em ambientes Linux. Desabilita o sandbox de segurança do Chrome.
  - `--disable-setuid-sandbox` : Desabilita o sandbox setuid.
  - `--disable-dev-shm-usage` : Desabilita o uso de `/dev/shm` . Pode ser útil em ambientes com recursos limitados.
  - `--disable-accelerated-2d-canvas` : Desabilita o canvas 2D acelerado por hardware.
  - `--disable-gpu` : Desabilita a aceleração de hardware da GPU. Útil em ambientes sem GPU ou para reduzir o consumo de recursos.
  - `--no-zygote` : Desabilita o processo zygote do Chrome, que pode ser útil em alguns ambientes.
  - `--disable-gl-drawing-for-tests` : Desabilita o desenho GL para testes.
  - `--disable-extensions` : Desabilita extensões do navegador.
  - `--disable-features=site-per-process` : Desabilita o isolamento de site por processo, o que pode reduzir o consumo de memória.
  - `--disable-web-security` : Desabilita a segurança web (use com cautela).
  - `--disable-features=IsolateOrigins,site-per-process` : Desabilita o isolamento de origem e

por processo.

- `--blink-settings=imagesEnabled=false` : Desabilita o carregamento de imagens (para web scraping onde imagens não são necessárias).
- `--autoplay-policy=no-user-gesture-required` : Desabilita a política de autoplay.
- `--disable-background-timer-throttling` : Desabilita a limitação de tempo de timers em segundo plano.
- `--disable-backgrounding-occluded-windows` : Desabilita o background de janelas ocluídas.
- `--disable-breakpad` : Desabilita o envio de relatórios de falhas.
- `--disable-client-side-phishing-detection` : Desabilita a detecção de phishing do lado do cliente.
- `--disable-component-extensions-with-background-pages` : Desabilita extensões de componente com páginas de fundo.
- `--disable-default-apps` : Desabilita aplicativos padrão.
- `--disable-features=TranslateUI` : Desabilita a interface de tradução.
- `--disable-hang-monitor` : Desabilita o monitor de travamento.
- `--disable-ipc-flooding-protection` : Desabilita a proteção contra flooding de IPC.
- `--disable-prompt-on-repost` : Desabilita o prompt em repost.
- `--disable-renderer-backgrounding` : Desabilita o background do renderizador.
- `--disable-sync` : Desabilita a sincronização.
- `--disable-domain-reliability` : Desabilita a confiabilidade de domínio.
- `--metrics-recording-only` : Grava apenas métricas.
- `--no-first-run` : Não executa a primeira execução.
- `--no-default-browser-check` : Não verifica o navegador padrão.
- `--mute-audio` : Silencia o áudio.
- `--enable-automation` : Habilita a automação.
- `--password-store=basic` : Define o armazenamento de senha como básico.
- `--use-fake-ui-for-media-stream` : Usa UI falsa para stream de mídia.
- `--use-fake-device-for-media-stream` : Usa dispositivo falso para stream de mídia.
- `--hide-scrollbars` : Esconde as barras de rolagem.
- `--ignore-certificate-errors` : Ignora erros de certificado.
- `--ignore-ssl-errors` : Ignora erros SSL.

    - `--window-size=1920,1080` : Define o tamanho da janela.
- **`userDataDir`** : Caminho para um diretório de dados do usuário. Usar um diretório de dados temporário ou limpo para cada execução pode evitar problemas de cache e cookies.
- **`timeout`** : Tempo máximo em milissegundos para esperar o navegador iniciar. Defina um valor adequado para evitar timeouts em ambientes lentos.
- **`ignoreHTTPSErrors`** : Se `true` , ignora erros de HTTPS/SSL. Útil para sites com certificados inválidos.

9. Opções de Página (`Page Options`)

As opções de página também são importantes para o desempenho:

- **`page.setDefaultNavigationTimeout(timeout)`** : Define o tempo limite padrão para operações de navegação na página. Aumente se as páginas demorarem a carregar.
- **`page.setViewport(viewport)`** : Define o tamanho da viewport da página. Um viewport menor pode reduzir o consumo de memória e CPU.
- **`page.setRequestInterception(true)`** : Habilita a interceptação de requisições para bloquear recursos desnecessários (imagens, CSS, fontes, etc.).
- **`page.setCacheEnabled(false)`** : Desabilita o cache do navegador para garantir que as páginas sejam sempre carregadas do servidor (útil para dados em tempo real).
- **`page.setJavaScriptEnabled(false)`** : Desabilita JavaScript se a página não depender dele para renderização de conteúdo estático.

10. Práticas Recomendadas Adicionais

- **Reutilização de Instâncias do Navegador:** Em vez de lançar um novo navegador para cada tarefa, reutilize uma única instância do navegador e abra novas páginas (`browser.newPage()`). Isso economiza tempo e recursos.
- **Fechamento de Páginas e Navegadores:** Sempre feche as páginas (`page.close()`) e as instâncias do navegador (`browser.close()`) quando não forem mais necessárias para liberar recursos.
- **Tratamento de Erros e Retentativas:** Implemente blocos `try-catch` e mecanismos de retentativa para lidar com erros de rede, timeouts e outros problemas, garantindo que o script não falhe inesperadamente.
- **Monitoramento de Recursos:** Monitore o uso de CPU e memória do processo do Puppeteer para identificar gargalos e otimizar o script. Ferramentas como Chrome DevTools e Lighthouse podem ajudar.

- **Otimização de Seletores:** Use seletores CSS eficientes e evite seletores complexos ou XPath quando possível.
- **Espera Inteligente:** Em vez de usar `page.waitForTimeout()`, que é ineficiente, use `page.waitForSelector()`, `page.waitForNavigation()` ou `page.waitForFunction()` para esperar por condições específicas na página.
- **Desabilitar Imagens e CSS:** Para web scraping, se você só precisa do texto, desabilitar o carregamento de imagens e CSS pode acelerar significativamente o processo e reduzir o consumo de largura de banda.

Ao aplicar essas configurações e práticas recomendadas, você pode otimizar significativamente o desempenho do Puppeteer para suas tarefas de automação e web scraping.