Ilustrando o impacto do uso de web workers na experiência do usuário em aplicacões web

Marco Antoni [1], Karina Wiechork[1], Andrea S. Charão[1]

[1] Programa de Pós-Graduação em Informática - Universidade Federal de Santa Maria
Santa Maria, RS, Brasil

{marco.antoni910, karinawiechork}@gmail.com, andrea@inf.ufsm.br

Abstract. *This article explores the use of web workers, which is one of the features introduced in the new HTML5 specifications, which allows solving some of the synchronous JavaScript problems. A prototype was developed with the purpose of illustrating how JavaScript can affect the user experience, with and without the use of web workers. We confirm that there are advantages to its use in order to keep the user interface always interactive, which ultimately improves the user experience.*

Resumo. *O presente artigo explora o uso de web workers, que é um dos recursos introduzidos nas novas especificações do HTML5, e que permite resolver alguns dos problemas do JavaScript síncrono. Foi desenvolvido um protótipo com o objetivo de ilustrar como o JavaScript pode afetar a experiência do usuário, com e sem o uso de web workers. Comprova-se que há vantagens na sua utilização de modo a manter a interface com o usuário sempre interativa, o que acaba melhorando a experiência do usuário.*

1. Introdução

O JavaScript é uma linguagem de programação executada no lado do cliente através de um navegador. Ela é interpretada e amplamente utilizada no desenvolvimento de aplicações web. Essa linguagem foi desenvolvida em 1995 por Brendan Eich e executada pela primeira vez no Netscape. É muito flexível e permite fazer desde a validação de formulários como também alterações na estrutura do HTML e do CSS, animações, carregamento de conteúdo dinâmico e até mesmo tarefas que exijam um grande poder de processamento [Zakas 2010]. Uma limitação da linguagem é o fato dela ter o fluxo de execução síncrono, que no caso de web sites pode ocasionar problemas de bloqueio da interface, impedindo a interação do usuário com a página. Também é comum em alguns casos ocorrer o travamento das abas do navegador e até mesmo ser apresentada uma mensagem para o usuário informando que a página parou de responder e perguntando se a página deve ser encerrada [Bidelman 2010].

Esses problemas acabam afetando negativamente a experiência do usuário, fazendo com que eles deixem de visitar a página e até deixem de efetuar compras em websites lentos causando prejuízos financeiros [Lopes 2018]. Felizmente, a linguagem tem evoluído muito nos últimos anos, onde foram introduzidos recursos que provém características assíncronas ao JavaScript, como por exemplo: *promises, async/await* e *web workers*.

O presente artigo está organizado da seguinte forma: a seção 2 apresenta o conceito de *web worker* e explica casos onde podem ser utilizados. Na seção 3, é apresentado o desenvolvimento de um protótipo onde é demonstrado o problema do

processamento síncrono do JavaScript. A seção 4 apresenta as conclusões obtidas e propõe trabalhos futuros explorando paralelismo com JavaScript.

2. Web Worker

O modelo de execução síncrono da linguagem impede que a página seja atualizada ou tenha interação com o usuário durante o período de execução de uma determinada tarefa incluindo a busca de outros arquivos JavaScript. A *Application Program Interface* - API *web worker* foi um dos recursos introduzidos junto com as especificações do HTML5, que permitem a execução de um script de forma concorrente devido ao fato dele ser executado em uma *thread* diferente da principal do navegador, deixando-a livre uma vez que é associada à interface [Bidelman 2010]. Apesar dessa API estar disponível desde 2010, muitos desenvolvedores não têm conhecimento do assunto.

Os *web workers* impõem algumas restrições de segurança que impedem a manipulação do DOM e o acesso a alguns metódos e propriedades do objeto *window*. Toda a comunicação se dá através da troca de mensagens por meio do método *postMessage()* onde a *thread* principal envia dados para o *worker* e vice-versa. Essas mensagens podem ser desde simples variáveis e até objetos do tipo JSON, porém é preciso ter cuidado pois a troca de dados ocorre por meio de cópia, que pode causar degradação de desempenho ao manipular grandes volumes de dados. Uma cópia por referência pode ser realizada através do objeto *ArrayBuffer*, porém a informação não fica mais acessível por quem a enviou, pois há uma transferência de contexto [Mozilla 2018].

Sua utilização é altamente recomendada em tarefas laboriosas ou em casos onde há necessidade de realizar uma pré busca de conteúdo, processamento de grandes arquivos ou respostas a requisições [Verdú e Pajuelo 2016]. Através dos *workers*, é possível atingir um certo grau de paralelismo nas aplicações, porém cabe ao desenvolvedor criar técnicas e algoritmos que possam tirar o máximo de proveito da API de modo a melhorar do desempenho da aplicação.

3. Implementação de um protótipo

Para demonstrar o impacto de *web workers* na experiência do usuário, foi construída uma aplicação que faz a ordenação de um *array* numérico, usando o algoritmo *insertionSort*. A interface oferece as opções de: definir a quantidade de itens a serem ordenados; a forma de geração do *array*; quantas vezes o teste deve ser repetido; visualizar o *array* gerado e dois botões que executam a tarefa na *thread* principal e o outro em *web workers* respectivamente. Ao lado direito da interface, há um *input* cujo objetivo é simular uma interação com o usuário, onde o botão "Enviar" executa uma requisição Ajax e escreve a resposta na tela. Para realização dos testes, foi definido um *array* de 100 mil registros aleatórios, no qual se pode constatar a demora na execução da tarefa e o uso intensivo da CPU, simulando um problema real de processamento com JavaScript.

Ao executar o processamento na *thread* principal, o usuário perde toda a interação com a página, devido aos fatos apresentados na seção 2, que acaba afetando sua experiência de navegação. A Figura 1 ilustra uma execução onde o usuário clica no botão responsável por realizar a requisição Ajax e a mesma só retorna a resposta após o término da execução do script.

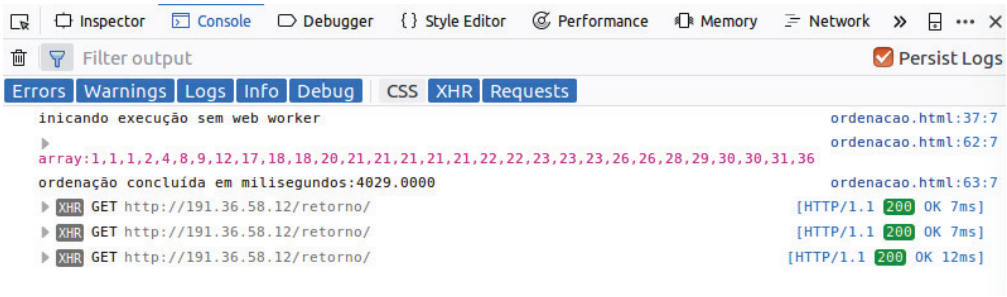

Figura 16: Código executado na thread principal

Na execução utilizando *web worker*, todo o processo de criação e ordenação do *array* é executado em segundo plano, deixando a *thread* responsável pela interface sempre ativa. A Figura 2 apresenta a interface da aplicação, onde é possível visualizar que houve interação do usuário mesmo durante a execução do scritpt, melhorando assim sua experiência.

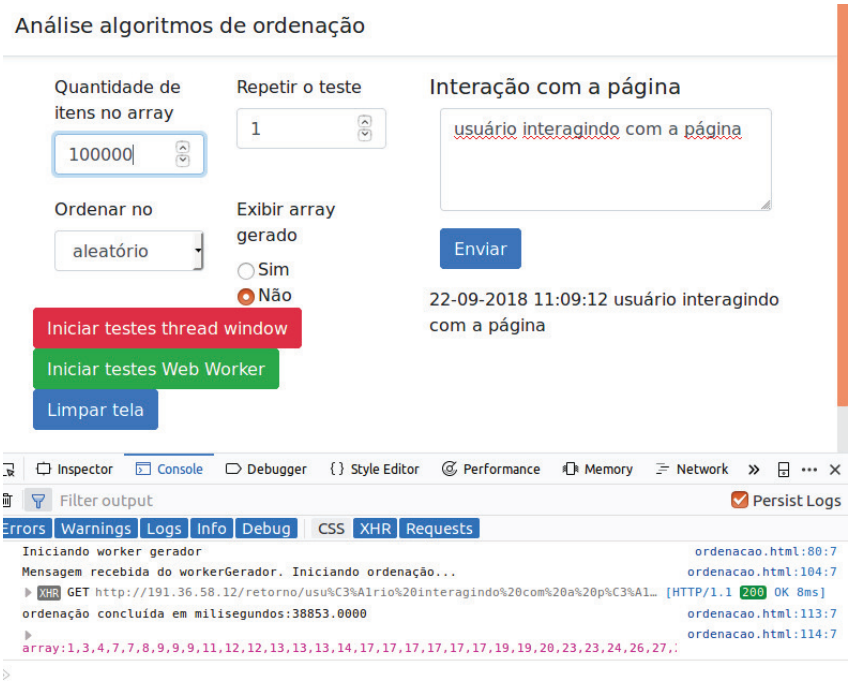

Figura 17: Processamento realizado em segundo plano

Em ambos os casos, percebe-se que não há ganho de desempenho na execução da tarefa, mas é visível que a experiência do usuário não é afetada, devendo o desenvolvedor usar técnicas que possam melhorar o desempenho da aplicação.

4. Conclusão e trabalhos futuros

Neste trabalho, exploramos o impacto da API *web worker* que foi introduzida através das novas especificações do HTML5, em que agora é possível executar tarefas de forma concorrente sem ocasionar bloqueios na *thread* principal do navegador. Apesar do recurso não ser novidade, uma parcela grande dos desenvolvedores ainda desconhece a tecnologia.

O desenvolvimento do protótipo teve por objetivo demonstrar a importância da utilização dos *web workers*. Apesar de não haver ganhos de desempenho, comprovou-se que há vantagens na sua utilização de modo a manter a interface com o usuário sempre interativa o que acaba melhorando sua experiência navegação.

Como trabalhos futuros, pretende-se adaptar outros algoritmos que são executados em *thread* única, para que o problema maior possa ser dividido em subproblemas e executá-los em vários *workers*, para descobrir até que ponto é possível ter ganho real de desempenho. Esse experimento deve ser repetido nos navegadores mais utilizados atualmente, a fim de identificar possíveis diferenças de desempenho para executar determinada tarefa.

Referências

Bidelman, E. (2010). O problema: simultaneidade do javascript. Disponível em: <https: //www.html5rocks.com/pt/tutorials/workers/basics/>. Acesso em: 20 de setembro de 2018.

Lopes, S. (2018). Tweetables: 14 fatos sobre performance web e otimizações. Disponível em: <http://sergiolopes.org/tweetables-performance-web-otimizacoes/>. Acesso em: 20 de setembro de 2018.

Mozilla (2018). Web workers API. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/API/Web_Workers_API>. Acesso em: 20 de setembro de 2018.

Verdú, J. e Pajuelo, A. (2016). Performance scalability analysis of javascript applications with web workers. IEEE Computer Architecture Letters, 15(2):105–108.

Zakas, N. C. (2010). JavaScript de Alto Desempenho. Editora Novatec, São Paulo.