УДК 004 **Информационные технологии**

Яровая Екатерина Владимировна, *магистранта Гродненского государственного университета им Я. Купалы, Беларусь, г. Гродно*

ОПТИМИЗАЦИЯ ПРИЛОЖЕНИЙ НА СТОРОНЕ КЛИЕНТА

Аннотация: В данной публикации рассмотрим основные процессы, происходящие в браузере после получения ответа в виде html документа, поговорим об основных этапах преобразования html разметки и взаимодействие с JavaScript. Также затронем механизмы, позволяющие оптимизировать или делегировать сложную и тяжелую со стороны процесса действия в веб воркир. Также рассмотрим принципы, реализованные в сервер воркерах, обсудим плюсы и минусы использования их. Кратко затронем молодую и мало используемую технологию как ворклет.

Ключевые слова: ворклет, веб воркер, оптимизация, браузер.

Annotation: In this publication we will study the main processes happening in a browser after receiving a response in the form of an html document, we will talk about the main stages of the transformation of html markup and JavaScript interaction. We also look at mechanisms that allow us to optimize or delegate complex and cumbersome processes to the web browser. We also look at the principles implemented in the server-workers, discuss the pros and cons of using them. We will touch briefly on a young and little used technology as a worklet.

Keywords: worklet, web worker, optimization, browser.

Для полного понимания давайте разберемся, что происходит в браузере, и как он вообще работает. Мы не будем обсуждать все процессы и потоки, которыми пользуются браузер. Мы посмотрим только на рендер процесс в котором есть какой-то механизм (движок), который отвечает за отображение

страницы пользователя.

Прежде чем он отобразит страницы он должен пройти ряд этапов. Первое что он делает после загрузки контента, он разбирает (парсинг) HTML и CSS в два дерева, следующий этап — это определить какое css свойство html элементу, это называется стайл (style) [1]. Затем проходит процесс определений, где эти элементы находятся и их размеры, лояут (layout). Далее начинается отрисовка, но тут есть особенность, отрисовывается не каждый отдельный элемент, а каждый композитный слой, композитные слои позволяют браузеру оптимизировать отрисовку элемента, суть заключается в том, что, если наш элемент потенциально может изменяться, его выносят в отдельный слой, где с ним можно работать. При работе JavaScript может вызываться любой из вышеперечисленных шагов или они могут пропускаться.

JavaScript также запускает движком рендеринга в Chrome, он называется Blink [3]. При запуске кода, движок обращается к платформе (Браузер), и к примеру, передает setTimeout, после того, как время таймаута закончится, браузер перенесет его в очередь, очередь обязана быть, так как язык однопоточный и не может выполнять задачи параллельно. Также в этом потоке находятся и все те шаги, которые мы обсудили выше, связанные с отрисовкой страницы. И это значит, что JavaScript мешает выполнению процессов отображения, а отображение препятствует выполнению JavaScript. Но для оптимизации процессов композит, который является достаточно тяжелой операцией, браузер может перенести в параллельный поток и выполнить всю работу в нем.

Какие решения могут быть, если мы хотим выполнить ресурсозатратную операцию в JavaScript, один из вариантов будет размять ее на несколько маленьких, чтобы не занимать основной поток. А на сколько маленькими они должны быть? Это зависит от того, что мы хотим отображать пользователю, допустим, анимация очень требовательная и мы должны придерживать 60FPS чтобы картинка была плавной.

С появлением web workers такая возможность есть. Это почти никак не

касается языка JavaScript, так что JavaScript, как был однопоточным языком, так и остался. Когда браузер выделяет один или несколько экземпляров движка JavaScript, каждый из экземпляров будет запускаться в отдельном потоке, и, следовательно, вы можете запускать разные сценарии в разных потоках, каждый из этих потоков и называется веб-работником (web workers). При создании нового воркера, мы ссылаемся на URL, где лежит файл с JavaScript, а не страницы HTML [2]. Тогда браузер запустит его в отдельном потоке, такой поток не пользуется ресурсами другого или главного процесса выполнения программы, что позволяет избежать всех проблем многопоточного программирования. А общаются они между собой с помощью передачи событий. Веб-воркеры могут также создавать воркеров, они будут называться сабворкер (subworkers), иногда это может быть полезно.

Во время работы воркера, это полностью изолированный процесс, но у нас есть ряд доступных и недоступных вещей, из доступных мы можем отметить: fetch api, webSocket, location, setTimeout, из недоступный это: DOM, local и session storage.

Для чего стоит использовать веб воркер: ресурсозатратные математические вычисления, сортировки огромных наборов данных, попиксельная обработка изображений или передача большого объема данных по сети. Для передачи данных есть несколько эффективных стратегий, для передачи алгоритма используют структурированный алгоритм клонирования. Еще один хороший или даже лучший вариант, это Transferable Objects, передаваемым объектом может быть просто типизированный массив [5].

Общение между работниками реализуется через общего работника (SharedWorker), у такого работника есть порты по аналогии с сетевыми портами, важно помнить, что при подключении к порту требуется инициализация, также обрабатывается событие – соединение (connect).

Веб воркеры - это отличное решение для определенных задач, но стоит отметить, что это весьма ресурсозатратно, нужно понимать, если приложение создает хотя бы один веб воркер, так как много ресурсов уходит на веб воркер,

приложение работает медленнее, если бы оно работало без веб воркера.

Осталось еще одна тема, которую хотелось бы упомянуть, она еще очень молодая, но очень перспективная и это Worklet [4]. Worklet - это облегченная версия веб воркеров, которая позволяет получить доступ к низкоуровневым частям процесса рендеринга. Основными особенностями являются ворклет, могут работать в любом из потоков, имеют несколько глобальных объектов, интересная особенность, они имеют произвольное время жизни, могут быть задействованы в разных JavaScript, имеют возможность работать в нескольких потоках одновременно, методы глобальной области видимости могут настраиваться и зависят от типа ворклета. Также он не настолько требователен к ресурсам. Очень немного информации о ворклетах, но хотелось бы упомянуть о проектах, которые делаются с помощью ворклетов, один из них должен позволять создавать свои лояуты (LayoutWorklet), paintWorklet для программной генерации изображения, AnimationWorklet предназначен для создания анимации с прокруткой и других высокозатратных процедер, AudioWorklet для обработки звука.

Библиографический список:

1. Пьюривал С., Основы разработки веб-приложений // Бестселлеры O'Reilly. 2015.

2. Шкляр Л., Архитектура веб-приложений // Информационные технологии. 2010.

3. Prateek Mehta, Creating Google Chrome Extensions // Найка о данных. 2016.

4. Tal Ater, Building Progressive Web Apps: Bringing the Power of Native to the Browser // Разработка. 2018.

5. Пабло Дилеман, изучаем Angular 2 // Дизайнер. 2022.