Анализ проблем последовательности загрузки клиентских веб-приложений

И.А. Короленко

Yandex Europe B.V., Amsterdam, Netherlands

Аннотация: В данной работе рассматриваются проблемы оптимизации загрузки клиентских веб-приложений и способы их решения с учетом различных практических условий. Приводятся способы ускорения загрузки веб-приложений и устранения блокирующих элементов в цепях обработки данных с целью улучшения различных аспектов пользовательского опыта. Предлагается подход, который позволяет спроектировать оптимальную цепь загрузки приложения, отвечающую высшим стандартам качества в front end индустрии и обеспечивающую наилучший опыт использования.
Ключевые слова: front end, рендеринг, клиентские веб-приложения, время загрузки, оптимизация производительности, пользовательский опыт.

Введение

При проектировании современных клиентских веб-приложений, одним из ключевых аспектов обеспечения должного качества конечного продукта является высокий уровень пользовательского опыта. Несмотря на то, что пользовательский опыт тяжело однозначно охарактеризовать и измерить с технической точки зрения, он имеет значительное влияние на желание пользователя продолжать использование приложения [1]. Стандарт ISO 9241 описывает пользовательский опыт, как восприятие и реакции пользователя, возникающие в результате использования и/или ожидаемого использования системы, продукта или услуги [2]. Пользовательский опыт в веб-приложениях складывается из множества компонентов, одним из наиболее важных в которых является начальное время загрузки приложения. Исследования Google показывают, что увеличение времени загрузки приложения с 1 до 3 секунд приводит к увеличению количества пользователей, принявших решение уйти на 32%, увеличение же этого показателя до 10 секунд приводит к увеличению оттока пользователей на 123% [3]. Также известно, что 53% пользователей выберут покинуть веб-

© Электронный научный журнал «Инженерный вестник Дона», 2007–2023

приложение, которое загружается дольше 3 секунд [4]. Данная статистика делает очевидной необходимость тщательной работы над скоростью и визуальными аспектами загрузки веб-приложений. Целью данной статьи является рассмотрение способов решения задачи оптимизации загрузки клиентских веб-приложений и методов устранения возникающих в процессе решения этой задачи проблем.

Критический путь рендеринга

Процесс загрузки веб-приложения является достаточно сложным, и наиболее важные его составляющие описываются понятием «критический путь рендеринга» (см. рис. 1) [5].

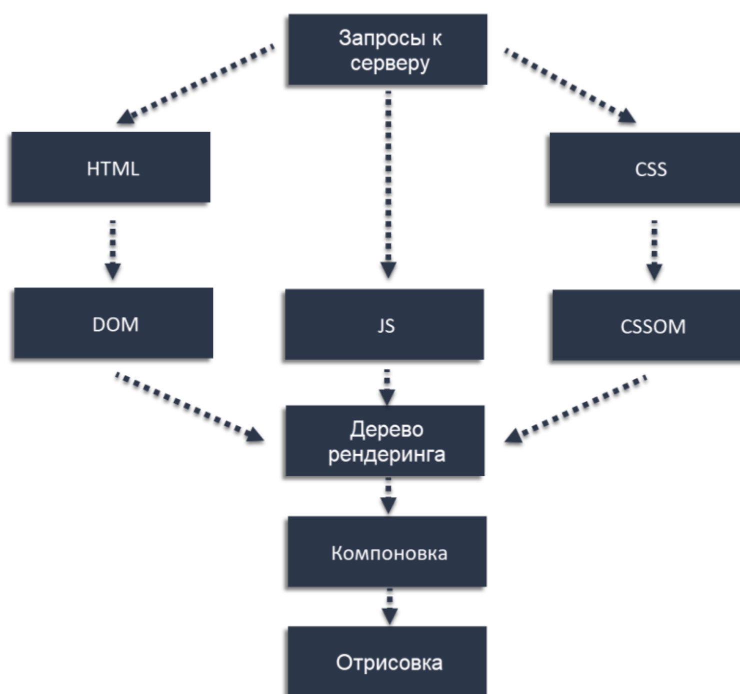

Рис. 1. – Критический путь рендеринга

Браузер отправляет запросы на сервер, осуществляет парсинг полученного от сервера HTML (HyperText Markup Language) и строит DOM-дерево (Document Object Model). Каждый раз, когда он попадает по ссылке на внешний ресурс (стили, скрипты, шрифты, изображения и т.д.), создается новый запрос. Некоторые запросы являются блокирующими, например, CSS

(Cascading Style Sheets), JS (JavaScript), шрифты, то есть приостанавливаются другие запросы до завершения работы над текущими. После завершения анализа DOM браузер создает CSSOM (CSS Object Model). CSSOM блокирует рендеринг до тех пор, пока не получит и не обработает все правила CSS. CSS-селекторы анализируются справа налево. Затем браузер создает дерево рендеринга, в котором рассчитывает стили для каждого видимого элемента на странице. Затем следует этап Компоновки, который определяет положение и размер элементов в дереве. Расположение зависит от размера экрана. На этом этапе определяется, где и как элементы будут расположены на странице и каковы отношения между элементами, в процессе могут произойти перекомпоновки. Например, если мы не указали размеры наших изображений. Затем идет этап Отрисовки. Страница фактически отображается для просмотра пользователем.

Метрики

Основными общепринятыми метриками для оценки скорости загрузки веб-приложения в веб-разработке являются так называемые Web Vitals. Web Vitals — это инициатива Google, направленная на предоставление единого руководства по качественным сигналам, которые необходимы для обеспечения комфортного взаимодействия с пользователем в Интернете [6]. Эти метрики помогают понять состояние «здоровья» приложения, то есть, насколько загрузка приложения отвечает ожиданиям пользователей. Проводить замеры можно в инструментах Lighthouse, Sentry, Datadog, и т.д.

Основные метрики Web Vitals:

- TTFB (Time to First byte) – время до первого байта, полученного от сервера. Хорошими являются значения менее 800 миллисекунд, приемлемыми – менее 1,8 секунд, значения выше 1,8 секунд считаются недопустимыми;

© Электронный научный журнал «Инженерный вестник Дона», 2007–2023

- FCP (First Contentful Paint) – время до отрисовки любого контента на странице. Хорошими являются значения менее 1,8 секунд, приемлемыми – менее 3 секунд, значения выше 3 секунд считаются недопустимыми;
- LCP (Largest Contentful Paint) – время до отрисовки большей части контента на странице. Хорошими являются значения менее 2,5 секунд, приемлемыми – менее 4 секунд, значения выше 4 секунд считаются недопустимыми;
- TTI (Time to Interactive) – время до момента, когда страница способна своевременно реагировать на действия пользователя. Иными словами, время для установки большинства обработчиков событий на странице. Хорошими являются значения менее 5 секунд, приемлемыми – менее 7,3 секунд, значения выше 7,3 секунд считаются недопустимыми;
- FID (First Input Delay) – время от первого пользовательского действия на странице до реакции на него. При измерении в Google Lighthouse следует использовать схожую метрику TBT (Total Blocking Time) ввиду отсутствия в этом инструменте метрики FID. Улучшение метрики TBT в большинстве случаев приводит к улучшению FID [7]. Хорошими являются значения менее 100 миллисекунд, приемлемыми – менее 300 миллисекунд, значения выше 300 миллисекунд считаются недопустимыми;
- CLS (Cumulative Layout Shift) – степень сдвига элементов страницы после первоначальной отрисовки. Хорошими являются значения менее 0,1 пунктов, приемлемыми – менее 0,25 пунктов, значения выше 0,25 пунктов считаются недопустимыми.

Проблемы цепи загрузки

К наиболее часто встречающимся проблемам цепи загрузки веб-приложений относятся нижеописанные.

1. Неоптимальное построение последовательности. Неправильная расстановка приоритетов ресурсов или неправильное понимание

© Электронный научный журнал «Инженерный вестник Дона», 2007–2023

последовательности Web Vitals. Корректная последовательность выглядит так: FCP -> LCP -> FID. Нам нужно соответствующим образом загрузить ресурсы. Пример: К моменту срабатывания LCP, JS-скрипты должны быть загружены, проанализированы и готовы к разблокировке взаимодействия (FID).

2. Загрузка сети/процессора. Ресурсы могут быть неправильно распределены для обеспечения полной загрузки ЦП (центральный процессор) и сети. Это приводит к «мертвому времени» ЦП, когда процесс ждет сеть, и наоборот. Примером этого являются скрипты, которые можно загружать одновременно или последовательно. Поскольку пропускная способность сети разделяется во время параллельной загрузки нескольких ресурсов, общее время загрузки всех ресурсов одинаково как для последовательной, так и для одновременной загрузки. Если вы загружаете ресурсы одновременно, процессор во время загрузки используется недостаточно, но при последовательной загрузке процессор может начать обработку ресурса, как только он будет загружен, это приводит к оптимальному использованию ЦП и сети.

3. Сторонние продукты. Сторонние библиотеки часто требуются для добавления в веб-приложение различного дополнительного функционала. Такие как реклама, аналитика, виджеты соц. сетей, чат, и другие встраиваемые элементы, как правило некритичные для работы приложения. Сторонняя библиотека, как правило, поставляется со своим собственным JS, изображениями, шрифтами и т.д. При этом сторонние продукты обычно нет стимула оптимизировать и поддерживать скорость загрузки сайта-потребителя. Такие компоненты могут добавлять большие затраты по времени, например, на выполнение JS кода, что задерживает время до интерактивности или мешает загрузке других важных ресурсов.

© Электронный научный журнал «Инженерный вестник Дона», 2007–2023

4. Специфичные для браузеров баги. При оптимизации последовательности загрузки нам в том числе необходимо учитывать баги, специфичные для конкретных браузеров. Пример: Баг предварительной загрузки в Chromium. Инструкцию preload можно использовать, чтобы указать браузеру загрузить ключевые ресурсы как можно скорее. Один из багов в Chromium приводит к тому, что запросы, отправленные через использование preload link, всегда запускаются раньше других запросов, даже если они имеют более высокий приоритет, что приводит к некорректной цепи загрузки [8].

Типичная неоптимизированная последовательность загрузки выглядит следующим образом. CSS ресурсы предварительно загружаются до JS-ресурсов, но не встроены в HTML. Загрузка сторонних JS-скриптов не контролируется и может блокировать рендеринг в любом месте документа. Шрифты не встроены в HTML и не используют preconnect. Шрифты загружаются через внешние стили, что задерживает загрузку. Главные изображения (изображения, занимающие большую часть страницы) не имеют приоритета. Изображения ATF (Above The Fold, т.е. отображаемые на первом экране приложения) и BTF (Below The Fold, т.е. отображаемые на втором и последующих экранах приложения) не оптимизированы.

Решение проблем цепи загрузки

Теперь давайте разберем возможные пути решений вышеописанных проблем. Первой рекомендацией будет использование паттерна PRPL (Push, Render, Pre-cache, Lazy) [9].

Суть PRPL паттерна заключается в следующем:

- приоритизация критически важных ресурсов с целью минимизации количества обращений к серверу и сокращающая время загрузки;
- рендеринг изначального маршрута в первую очередь для улучшения пользовательского опыта;

© Электронный научный журнал «Инженерный вестник Дона», 2007–2023

- предварительное кэширование ресурсов в фоновом режиме для часто посещаемых маршрутов, чтобы минимизировать количество запросов к серверу и улучшить работу в автономном режиме;
- ленивая загрузка маршрутов и ресурсов, которые не запрашиваются часто.

Минимизируйте количество критически важных ресурсов, отложив загрузку большинства, пометив их, как async/defer и сгруппировав. Оптимизируйте количество требуемых запросов и размер файлов. Оптимизируйте порядок загрузки, чтобы критические ресурсы были загружены первыми, что сокращает продолжительность критического пути рендеринга. Используйте кэширование браузера и сжатие с алгоритмом gzip или Brotli. Алгоритм Brotli, как правило, дает лучшую степень сжатия при прочих равных [10]. Независимо от алгоритма сжатия, существует общее правило:

$$compress(a+b) <= compress(a) + compress(b),$$

т.е. один большой пакет обеспечит лучшее сжатие, чем несколько меньших. Используйте CDN (Content Delivery Network), чтобы воспользоваться преимуществами периферийного кэширования.

CSS и шрифты должны загружаться с наивысшей приоритизацией. Критический CSS (минимальный для FCP) должен быть встроен в HTML, изображения должны иметь установленный размер, а JS должен быть разделен на несколько файлов и загружен постепенно (например, через динамический импорт). При разделении JS-ресурсов нам необходимо соблюдать компромисс между гранулярностью и производительностью: при более высокой гранулярности для отдельного маршрута требуется меньше JS, и мы можем кэшировать общие зависимости, однако слишком больше количество маленьких фрагментов снижает степень сжатия для отдельных фрагментов и негативно влияет на производительность браузера. В случае,

© Электронный научный журнал «Инженерный вестник Дона», 2007–2023

если встраивание невозможно, критический CSS должен быть предварительно загружен и использован из того же источника, что и документ. Следует избегать использования критически важного CSS из нескольких доменов или использования стороннего критического CSS, такого как Google Fonts. Вместо этого ваш сервер может служить прокси-сервером для стороннего критического CSS. Слишком много встроенного CSS может привести к раздуванию HTML и длительному анализу стилей в основном потоке. Это может навредить FCP, плюс, встроенный CSS нельзя кэшировать.

Критические шрифты. Так же, как и критический CSS, CSS для критических шрифтов тоже должен быть встроенным. Если встраивание невозможно, скрипт следует загрузить с использованием preconnect. Задержка при получении шрифтов из другого домена может повлиять на FCP. Preconnect сообщает браузеру, что ему необходимо заранее подключиться к этим ресурсам. Встроенные шрифты могут значительно увеличить размер HTML и задержать начало получения других важных ресурсов. Резервный шрифт можно использовать, чтобы разблокировать FCP и сделать текст доступным. Однако, использование резервного шрифта, может повлиять на CLS из-за процесса замены. Это также может повлиять на FID из-за потенциально большой задачи замены стилей и повторной компоновки в основном потоке при установке основного шрифта. Загружайте ATF-изображения как можно скорее, и применяйте ленивую загрузку для BTF изображений.

Основной JS критически влияет на готовность приложения к взаимодействию с пользователем. Его загрузка может задерживаться из-за загрузки изображений и стороннего JS, сторонний JS также может блокировать основной JS в основном потоке. Следовательно, основной JS должен быть загружен раньше ATF-изображений и начать выполняться

© Электронный научный журнал «Инженерный вестник Дона», 2007–2023

раньше стороннего JS в основном потоке. Также стоит учесть, что основной JS не блокирует FCP и LCP на страницах, которые рендерятся на стороне сервера. Сторонний JS в head-части HTML может блокировать синтаксический анализ CSS и шрифтов и, следовательно, увеличивать время FCP. Скрипт в заголовке также блокирует анализ тела HTML. Выполнение стороннего JS в основном потоке может задержать гидратацию (в случае рендеринга на стороне сервера) и FID. Поэтому загрузку стороннего JS стоит тщательно оптимизировать, используя async, defer, preconnect и dns-prefetch, использовать ленивую загрузку и self-hosting, а также использовать сервис-воркеры для кэширования ресурсов. Скрипты получают разные приоритеты, в зависимости от того, где они находятся в документе и являются ли они асинхронными, отложенными или блокирующими. Блокирующие скрипты, запрошенные перед первым изображением, имеют более высокий приоритет по сравнению с блокирующими скриптами, запрошенными после получения первого изображения. Асинхронные/отложенные/встроенные скрипты, независимо от того, где они находятся в документе, имеют самый низкий приоритет. Таким образом, мы можем расставить приоритеты для различных ресурсов, используя соответствующие атрибуты.

Заключение

При проектировании клиентских веб-приложений особое внимание стоит уделять оптимизации цепи загрузки, так как результат этого процесса во многом определит качество пользовательского опыта и вытекающие из этого показатели эффективности выполнения поставленных перед приложением целей. Рассмотрены основные метрики определения эффективности процесса загрузки, а также наиболее распространенные проблемы, возникающие в этом аспекте при проектировании клиентских веб-приложений. Приведены ключевые способы решения рассмотренных проблем. Предложенный подход поможет спроектировать оптимальную цепь

© Электронный научный журнал «Инженерный вестник Дона», 2007–2023

загрузки приложения, отвечающую высшим стандартам качества в front end индустрии и обеспечивающую наилучший опыт использования.

Литература (References)

1. Kuniavsky M. Observing the User Experience: A Practitioner's Guide to User Research. Burlington, Massachusetts: Elsevier Science, 2012. 608 p.

2. Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems. Geneva, Switzerland: ISO, 2019. 33 p.

3. An D. Find out how you stack up to new industry benchmarks for mobile page speed. Think with Google, 2018. URL: thinkwithgoogle.com/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks/.

4. Google Consumer Insights. Think with Google, 2023. URL: thinkwithgoogle.com/consumer-insights/consumer-trends/mobile-site-load-time-statistics/.

5. Shroff P., Chaudhary S. Critical rendering path optimizations to reduce the web page loading time. IEEE, 2017. URL: ieeexplore.ieee.org/document/8226266.

6. Walton P. Web Vitals. web.dev, 2020. URL: web.dev/articles/vitals?hl=en.

7. Walton P. Total Blocking Time (TBT). web.dev, 2019. URL: web.dev/articles/tbt?hl=en.

8. Hallie L., Osmani A. Optimize your loading sequence. patterns.dev, 2023. URL: patterns.dev/posts/loading-sequence.

9. Hallie L., Osmani A. PRPL Pattern. patterns.dev, 2023. URL: patterns.dev/posts/prpl.

10. Szabadka Z. Introducing Brotli: a new compression algorithm for the internet. Google Open Source, 2015. URL: opensource.googleblog.com/2015/09/introducing-brotli-new-compression.html.

© Электронный научный журнал «Инженерный вестник Дона», 2007–2023