МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

Тема: Описание предполагаемого способа решения

Студентка гр. 4304 Лапцевич Д.А.

Санкт-Петербург

2019

СОДЕРЖАНИЕ

1. АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ ВЕБ-ПРИЛОЖЕНИЙ

1.1. Показатели производительности веб-приложений

Исследование производительности веб-приложение можно проводить по следующим показателям:

1. время на редирект;
2. время ответа сервера;
3. время до первого байта;
4. загрузка DOM (содержимого страницы - контента);
5. время до интерактива (когда пользователь может начать взаимодействовать со страницей);
6. время полной загрузки.

Когда выполняется запрос для страницы, Front-end и Server-side компоненты затрачивают определенное количество времени для выполнения своих операций. Поскольку эти операции по существу последовательны, их суммарное время можно считать общим временем загрузки страницы.

1.2. Стоимость JavaScript

Поскольку сейчас пишутся сайты, которые в большей степени зависят от JavaScript, приходится платить за данные, которые отправляются на сайт, но это не всегда можно легко заметить. Поэтому очень важно, чтобы сайт загружался и быстро работал на мобильных устройствах.

Когда большинство разработчиков думают о стоимости JavaScript, они думают об этом с точки зрения стоимости загрузки и исполнения. Отправка большего количества байтов JavaScript по кабелю занимает тем больше времени, чем медленнее соединение пользователя [1].

1.2.1 Загрузка JS

Снизить стоимость передачи JavaScript по сети можно следующими способами (рисунок 1):

- отправление только того кода, который нужен пользователю. В данной ситуации может быть очень полезно разделение кода;
- минимизация (Uglify для ES5, babel-minify или uglify-es для ES2015);
- сильное сжатие (с использованием Brotli ~ q11, Zopfli или gzip). Brotli превосходит gzip по степени сжатия. Компании CertSimple удалось сократить на 17% JS файлы, а LinkedIn сэкономить 4% на времени загрузки;
- удаление неиспользуемого кода. Определить неиспользуемый код можно с помощью DevTools code coverage. Для удаления неиспользуемого кода можно использовать tree-shaking, Closure Compiler's и библиотеки, такие как lodash-babel-plugin или ContextReplacementPlugin Webpack для библиотек, таких как Moment.js. Также рекомендуется использование babel-preset-env & browserlist в современных браузерах. Также необходимо проводить анализ Webpack bundle для выявления возможностей удалить неиспользуемые зависимости.

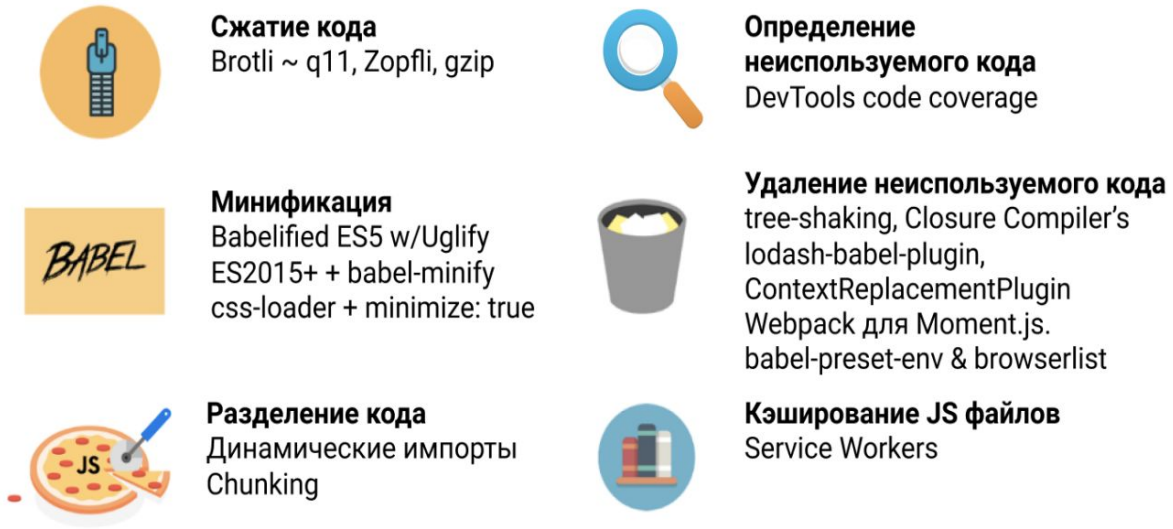

Рис. 1. Рекомендации по сокращению количества JavaScript

Кэширование JS файлов, для минимизирования количества сетевых запросов. Также необходимо определить оптимальные сроки жизни для скриптов (max-age) и токенов проверки поставки (ETag), чтобы избежать передачи неизмененных байтов. Кэширование с помощью Service Worker'ов может сделать сеть приложений устойчивой и предоставит быстрый доступ к таким функциям, как кеш-код V8.

1.2.2. Парсинг / Компиляция

После загрузки одно из самых больших - это время, когда JS-движок парсит / компилирует JavaScript код.

Большое время парсинга / компиляции кода могут сильно влиять на то, как скоро пользователь сможет взаимодействовать с сайтом. Чем больше JavaScript кода отправляется, тем больше времени необходимо для его парсинга и компиляция, то есть до момента, когда сайт станет интерактивным.

В сравнении с JavaScript также множество затрат на обработку изображений с эквивалентным размером (так как их еще нужно декодировать), но в среднем требуется больше времени для обработки javascript кода, чем для загрузки изображений.

Очень важно понимать, что байты JavaScript и байты изображения имеют очень разные затраты. Обычно изображения не блокируют основной поток или не препятствуют взаимодействию с интерфейсом при декодировании и растрировании. Однако JS может задерживать интерактивность из-за временных затрат на парсинг, компиляцию и исполнение.

Долгий парсинг и компиляции очень важны, когда речь идет о средних мобильных телефонах. В среднем у пользователей могут быть телефоны с медленными процессорами и графическими процессорами, без кеша L2 / L3, которые также могут иметь ограниченную память.

1.2.3. Время выполнения

Это не просто парсинг и компиляция кода, которые влияют на стоимость. Выполнение JavaScript (запуск кода парсинг / компиляции) является одной из операций, которая выполняется в основном потоке. Длительное время выполнения также влияет на то, как скоро пользователь может взаимодействовать с сайтом.

Чтобы решить эту проблему, необходимо разбивать JavaScript код на небольшие фрагменты, чтобы избежать блокировки основного потока.

Для того, чтобы сохранить время парсинга/компиляции и времени передачи JavaScript кода, существуют паттерны, которые могут помочь в решении этих задач, например, route-based chunking или PRPL.

1.2.4. Другие расходы

JavaScript может влиять на производительность страницы другими способами:

- Утечка памяти. Страницы могут часто приостанавливать свою работу, то есть терять свою интерактивность из-за сборщика мусора (garbage collector), что очень заметно в быстро реагирующих приложениях. Основным недостатком сборщиков мусора является недетерминированность, то есть сложно понять в какой момент может произойти сборка мусора. Когда браузер восстанавливает память, выполнение JS приостанавливается, поэтому браузер, часто собирающий мусор, может приостанавливать выполнение чаще, чем может быть удобно пользователям. Необходимо избегать утечек памяти и частых пауз сборщиков мусора, чтобы страницы не теряли свою интерактивность. Существует четыре самых распространенных вида утечек памяти JavaScript: случайные глобальные переменные, забытые коллбэки и таймеры, ссылки на удаленные из DOM элементы, замыкания.

- Во время запуска, JavaScript с долгим временем выполнения кода может блокировать основной поток, делая недоступными страницы. Разбивка кода на более мелкие части (с использованием requestAnimationFrame() или requestIdleCallback() для планирования) может минимизировать проблемы с откликом сайта.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Цена JavaScript. URL: https://habr.com/en/post/343562/ (дата обращения: 20.12.2019).
2. Garbage collector. URL: https://habrahabr.ru/post/309318/ (дата обращения: 20.12.2019).