DOI 10.37882/2223-2966.2024.7-2.29

ОПТИМИЗАЦИЯ КРОССБРАУЗЕРНОЙ ЭФФЕКТИВНОСТИ ВЫСОКОПОСЕЩАЕМЫХ ВЕБ-ПРИЛОЖЕНИЙ

OPTIMIZATION OF CROSS-BROWSER EFFICIENCY OF HIGHLY VISITED WEB APPLICATIONS

A. Prisyazhnyy

Summary. The constantly evolving landscape of web technologies and the need to support older browsers for accessibility to all possible users poses serious challenges for the engineering community to provide universal support in various browsers. Given that not all modern Internet users access the Internet from the latest browsers, applications may have problems with cross-browser compatibility. Experimental data suggest that the ES6 specification of the JavaScript language, published in 2015, on average offers better execution speed in modern browsers compared to its predecessors, but not all browsers used natively support it. This article examines different ways to achieve a compromise between optimal site performance and cross-browser support. The value of the presented work lies in the fact that the author offers a unique practical guide on configuring the transpilation of a web application to achieve maximum cross-browser compatibility and support for outdated browsers.

Keywords: optimization of web applications, transpilation, cross-browser compatibility, support for outdated browsers.

Присяжный Александр Олегович
Старший инженер-программист,
Челябинский Государственный Университет;
ООО «Тасит Ноледж Латин Америка»
a.prisazhny@yandex.ru

Аннотация. Постоянно развивающийся ландшафт веб-технологий и необходимость поддерживать старые браузеры для доступности всем возможным пользователям ставит инженерному сообществу серьезные задачи по обеспечению универсальной поддержки в различных браузерах. Учитывая, что не все современные интернет-пользователи выходят в интернет с новейших браузеров, у приложений могут возникать проблемы кроссбраузерной совместимости. Экспериментальные данные говорят о том, что изданная в 2015 году спецификация ES6 языка JavaScript в среднем предлагает лучшую скорость выполнения в современных браузерах по сравнению с её предшественниками, но не все используемые браузеры нативно ее поддерживают. Данная статья рассматривает разные способы достижения компромисса между оптимальной производительностью сайта и кроссбраузерной поддержкой. Ценность представленной работы состоит в том, что автором предлагается уникальное практическое руководство по настройке транспиляции веб-приложения для достижения максимальной кроссбраузерной совместимости и поддержки устаревших браузеров.

Ключевые слова: оптимизация веб-приложений, транспиляция, кроссбраузерная совместимость, поддержка устаревших браузеров.

Введение

С развитием веб-технологий стало очевидно, что поддержка веб-приложений представляет собой одну из наиболее сложных задач в области разработки. В отличие от стандартных приложений, предназначенных для конкретных операционных систем, веб-приложения должны быть доступными для широкого круга пользователей независимо от их используемой операционной системы или браузера.

Традиционные приложения, устанавливаемые на компьютеры или мобильные устройства, часто разрабатываются с учетом конкретной операционной системы, что делает их оптимизацию и тестирование относительно предсказуемыми. Веб-приложения же сталкиваются с гораздо большим разнообразием переменных, таких как разные версии браузеров, различные разрешения экранов и разнообразные настройки пользовательских устройств. Эти особенности делают задачу поддержки веб-приложений особенно актуальной и сложной.

Основное предназначение веб-приложений — предоставление одинакового уровня сервиса для всех пользователей. Это означает, что разработчики должны уделять особое внимание тому, чтобы приложение было не только функциональным, но и корректно работало в разных браузерах и на разных устройствах. Современные средства автоматизации сборки позволяют решать эту проблему с помощью полифилов и транспиляции, но в результате мы получаем несколько более тяжелую и медленную в исполнении сборку.

Результаты и обсуждение

Архитектура браузеров и совместимость.

На текущий момент рынок браузеров проходит этап доминирования браузера Google Chrome с его JavaScript-движком V8 [1, с. 232]. Этот факт облегчает поддержку для разработчиков, так как большинство современных браузеров, а также Node.js — универсальная платформа для JavaScript — строится на основе этого движка, что обеспечивает высокую степень совместимости JavaScript-приложений.

Несмотря на преобладание браузеров на основе Chromium среди современных браузеров, некоторые известные браузеры используют собственные, уникаль-

ные движки для обработки и отображения веб-контента [3]. Эти движки могут интерпретировать и выполнять код по-разному, что порой приводит к различиям в отображении и функционировании веб-сайтов. Разработчикам важно понимать эти различия и учитывать их при создании кросс-браузерных приложений.

Одним из ключевых преимуществ большинства браузеров является их обратная совместимость [4]. Это означает, что если определенная функция языка поддержана в определенной версии браузера, то она будет поддерживаться и в последующих версиях. Такой подход несколько упрощает задачу разработчикам, но требует также знания об устаревших функциях и технологиях.

Использование старых версий JavaScript может казаться привлекательным решением для достижения максимальной совместимости, однако такой подход не лишен недостатков. С одной стороны, это гарантирует работу приложения на большинстве устройств, но с другой стороны, это может привести к увеличению объема кода, повышению времени загрузки, увеличению расходов мобильного трафика, замедлению работы приложения и ухудшению пользовательского опыта.

Методы обеспечения совместимости.

ES6 — это спецификация языка JavaScript 2015 года, вторая большая его ревизия — которая в среднем предлагает более быструю скорость в современных браузерах [5], которая представила множество современных функций в языке [6] и поддерживается подавляющим большинством используемых браузеров.

JavaScript спецификации ES6 полностью поддерживается 95,4 % по глобальной статистике browsersl. ist [7, с. 212]. Также в разделе preset-env документации транспилятора Babel можно посмотреть какие именно функции языка поддерживаются в разных его спецификациях, откуда видно, что спецификация ES6 привнесла наибольшее количество функций в язык JavaScript в сравнении со всеми остальными, поэтому, с учетом глобальной статистики использования браузеров с поддержкой ES6, разделение поддержки веб-приложения по этой спецификации имеет наибольший смысл.

— Первый метод обеспечения совместимости заключается в полной транспиляции приложения в устаревшую версию JavaScript. Это можно грубо сравнить с установкой приложения, предназначенного для старой версии операционной системы, на новое устройство. Хотя такой подход обеспечивает более широкую совместимость, и поэтому является выбором большинства веб-приложений с большим посещением, он имеет определенные недостатки, такие как увеличенный объем кода и замедленное выполнение;

— Второй метод не вполне помогает достичь универсальной совместимости, но это один из путей, который выбирают многие команды, разрабатывающие веб-приложения. Он заключается в полном отказе от поддержки устаревших браузеров. Это может быть обосновано статистикой, где большинство выходят в сеть через браузеры современных версий, нативно поддерживающих ES6. На момент написания статьи это 95,4% по глобальной статистике browsersl.ist [7, с. 211]. Локальная статистика может отличаться. Например, browsersl.ist уведомляет что в России спецификация ES6 поддерживается лишь 80,3% процентами посетителей, т. е. для каждого пятого пользователя региона, веб-приложение будет работать некорректно [8]. Такой компромисс для веб-приложений с высокой посещаемостью недопустим, поскольку лишает их заметной доли интернет-трафика и, как следствие, предположительно столь же заметной доли прибыли.

Такой подход может исключить определенную часть пользовательской аудитории, особенно в регионах с большей долей использования устаревших браузеров или в корпоративном секторе, где корпоративный регламент диктует использование браузера с устаревшей версией движка JavaScript;

— Третий метод был сформулирован в ходе исследования. Он, возможно, самый оптимальный, но требует больше работы со стороны разработчиков. Он включает в себя сборку двух разных пакетов кода: одного для современных браузеров и другого для устаревших. В зависимости от браузера пользователя загружается и исполняется соответствующий пакет. Это обеспечивает быстродействие и современные возможности для большинства пользователей, а также гарантирует работоспособность сайта для тех, кто использует старые браузеры.

Из недостатков этого метода следует отметить, что в устаревших браузерах запустится только устаревшая сборка, но загружена будет и устаревшая, и современная. Этот компромисс заденет небольшую часть аудитории, предоставив подавляющему большинству пользователей лучший опыт в сравнении с другими подходами, сохраняя поддержку всех пользователей.

Третий метод — создание двойной сборки — является оптимальным для веб-приложений с высокой посещаемостью, поскольку предлагает современным браузерам легкую и современную сборку приложения, при этом, поддерживая устаревшие браузеры сборкой с устаревшим же JavaScript.

Описание конфигурации.

В ходе исследования был выявлен способ достижения такой конфигурации, основывающийся на нескольких инструментах, повсеместно используемых в современных веб-приложениях:

— Сборщик Webpack;
— Транспилятор Babel;
— Файл управления версиями и регистрации зависимостей проекта `package.json`;
— Webpack-плагин `HtmlWebpackPlugin`;
— Встроенная в браузер поддержка разделения современных и устаревших скриптов документах.

Поскольку большинство современных веб-приложений используют одинаковые инструменты создания и сборки проекта, предположим, что проект инициирован с помощью NPM и что он уже использует Webpack, HtmlWebpackPlugin и Babel для создания сборки [9, с. 117]. Для того чтобы использовать Webpack и Babel для создания двух различных сборок — одну для современных браузеров и другую для старых браузеров — необходимо следовать нижеуказанным шагам. На рис. 1 схематично изображен процесс создания двойной сборки.

1. В файле `webpack.config.js` получить доступ к аргументам, считывая параметр argv в функции, возвращающей конфигурацию проекта. Это пригодится далее для проверок на то, какая версия сборки запущена в конкретный момент.
2. Ниже, в поле конфигурации `output`, означающем куда Webpack будет выводить сборки, проверить, равно ли поле argv.mode строке `modern`. Если равно, значит это современная сборка, задать имя файла современной сборки (например, `modern.js`), а если нет — указать имя устарев-шей сборки (например, `legacy.js`). Этот шаг определяет название и расположение соответствующего варианта сборки.
3. Далее, в разделе загрузчиков, среди загрузчиков для файлов с расширением `js` найти `babel-loader` и сконфигурируйте для него пресет `@babel/preset-env` таким образом, чтобы там было поле `target`, а его значение снова зависело от поля argv.mode. Если оно равно строке `modern`, то передаем значение ` {esmodules: true} `, что гарантирует, что Babel будет транспилировать код для современных браузеров, которые поддерживают ES6-модули. Если не равно — передаем ` {ie: "11»}`, чтобы включить в сборку функции, которые поддерживались бы устаревши-ми браузерами.
4. В файле `package.json` добавить три новых скрипта: “build:modern” со значением `webpack --mode modern` и `build:legacy` со значением `webpack --mode legacy`. Этот шаг нужен для разделения процессов современной и устаревшей сборки. Именно здесь мы определяем аргумент mode, на который мы опираемся для определения имени файла сборки и определения состава сборки в предыдущих трёх шагах. Также добавить скрипт `build`, который последовательно запустит другие два — `npm run build:modern && npm run build:legacy`.
5. В итоге требуется добавить полученные файлы в `index.html`. Для этого, либо вручную добавьте туда строки ` <script type="module" src="modern.js"></script>` и `<script nomodule src="legacy.js"></script>`, либо воспользуйтесь шаблонизацией для динамического добавления скриптов в документ как на рис. 2.
6. Помимо прочего, браузер Safari версии 10.1 поддерживает атрибут `type="module"`, но не поддерживает `nomodule`. Для того, чтобы исправить это, потребуется также добавить в документ полифил показанный на рисунке 3.

Для обеспечения максимальной совместимости я воспользовался сочетанием двух сборок. Несмотря на некоторые недостатки, такой подход обеспечивает оптимальный пользовательский опыт для подавляющего большинства посетителей веб-приложения.

При желании можно построить больше версий сборок для разных поколений браузеров и на сервере, анализируя значение заголовка User-Agent, возвращать соответствующую сборку, поскольку браузеры не имеют встроенной возможности работать с разными версиями сборки помимо `type="module"` и `nomodule`.

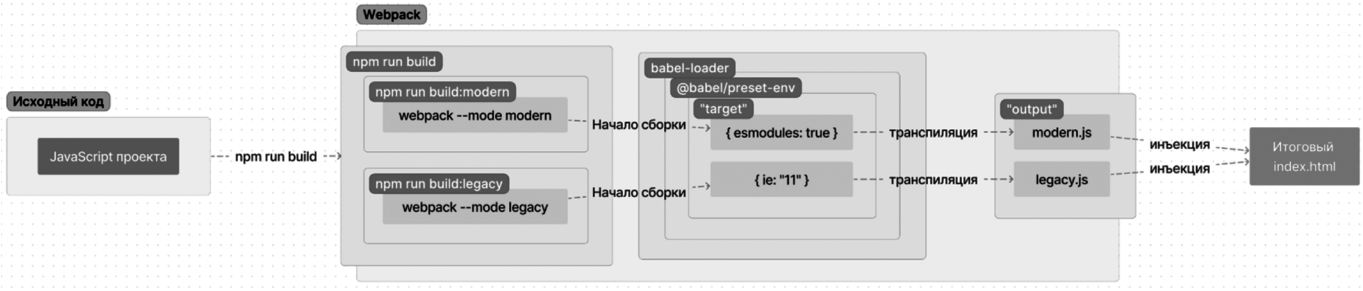

Рис. 1. Схема процесса создания двойной сборки

```
<!-- Начало HTML-документа -->

<%= htmlWebpackPlugin.files.js.map(js => {
  if (js.endsWith('modern.js')) {
    return `<script type="module" src="${js}"></script>`;
  } else if (js.endsWith('legacy.js')) {
    return `<script nomodule src="${js}"></script>`;
  }
  return '';
}).join('') %>

<!-- Конец HTML-документа -->
```

Рис. 2. Шаблонизация для динамического добавления скриптов

```
<script type="module">
(function() {
  var check = document.createElement('script');
  if (!('noModule' in check) && 'onbeforeload' in check) {
    var support = false;
    document.addEventListener('beforeload', function(e) {
      if (e.target === check) {
        support = true;
      } else if (!e.target.hasAttribute('nomodule') || !support){
        return;
      }
      e.preventDefault();
    }, true);

    check.type = 'module';
    check.src = '.';
    document.head.appendChild(check);
    check.remove();
  }
}());
</script>
```

Рис. 3. Полифил для поддержки nomodule в Safari 10.1

Заключение

В современной веб-разработке обеспечение совместимости с различными браузерами является одним из основных вызовов. Спецификация ES6 внесла множество современных функций в язык JavaScript, однако не все браузеры ее поддерживают. В этой статье были рассмотрены различные методы обеспечения совместимости: полная транспиляция, отказ от поддержки устаревших браузеров и создание двух отдельных сборок для современных и устаревших браузеров. Исследование показало, что третий метод, хотя и требует больше усилий со стороны разработчиков, предоставляет оптимальное сочетание производительности и совместимости. Практическое руководство, представленное в данной статье, предоставляет возможность получить следующие преимущества при разработке приложений:

— широкий охват аудитории. Разные люди предпочитают различные браузеры и платформы, поэтому оптимизированный веб-сайт будет доступен для большего количества пользователей;
— улучшенный пользовательский опыт. Когда веб-сайт или приложение работает надежно и отображается правильно на любом браузере, пользователи чувствуют себя комфортно и удовлетворены использованием продукта;
— увеличение продаж и конверсий. Если пользователи могут легко просматривать и покупать товары на вашем веб-сайте независимо от используемого ими браузера, это может привести к большему количеству продаж и увеличению конверсий;
— улучшение репутации бренда. Если веб-сайт отображается и работает хорошо на различных платформах и браузерах, это создает положительное впечатление о вашей компании и может укрепить репутацию бренда;

— удобство для разработчиков. Оптимизация кроссбраузерной эффективности может также упростить разработку и поддержку веб-сайтов и приложений, так как разработчику необходимо будет уделять меньше времени исправлению ошибок и приведению кода в соответствие с различными браузерами. В целом, оптимизация кроссбраузерной эффективности позволяет достичь более высокой производительности и удовлетворенности пользователей, что, в конечном счете, может привести к увеличению прибыли и успеху вашего веб-сайта или приложения.

Используя комбинацию инструментов, таких как Webpack, Babel, и HtmlWebpackPlugin, разработчики могут эффективно строить и поддерживать обе версии своих приложений. Помимо прочего, была учтена особенность нативной поддержки такого разделения браузером Safari 10.1 и применено решения для обеспечения и его поддержки. Таким образом, несмотря на некоторые сложности, данная стратегия представляет собой оптимальный выбор для большинства современных веб-приложений с высокой посещаемостью, стремящихся к предоставлению лучшего пользовательского опыта посетителям с любыми известными браузерами.

ЛИТЕРАТУРА

1. Варданян В.Г. Методы оптимизации программ на языке JavaScript, основанные на статистике выполнения программы / В.Г. Варданян // Труды Института системного программирования РАН. — 2016. — Т. 28, № 1. — С. 5–20. — DOI 10.15514/ISPRAS-2016–28(1)–1. — EDN WAPNKT.
2. Движок JavaScript / [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Движок_JavaScript (дата обращения: 12.09.2023).
3. Денисова Н.Н. Оптимизация информационной системы «Аргументатор стоимости» / Н.Н. Денисова, Е.П. Мельник, Т.В. Тюпикова // Анализ, моделирование, управление, развитие социально-экономических систем (АМУР-2022): сборник научных трудов XVI Международной школы-симпозиума АМУР-2022, Симферополь-Судак, 14–27 сентября 2022 года. — Симферополь: Индивидуальный предприниматель Корниенко Андрей Анатольевич, 2022. — С. 139–144. — EDN PQLMWY.
4. Кайл Симпсон. You Don't Know JS / [Электронный ресурс]. URL: https://github.com/getify/You-Dont-Know-JS/blob/2nd-ed/get-started/ch1.md#backwards--forwards (дата обращения: 12.09.2023).
5. Красильников И.С. Методы оптимизации и повышения скорости работы клиентской части веб-приложения / И.С. Красильников // Научные труды магистрантов и аспирантов: Сборник научных трудов / Отв. редактор Д.А. Погонышев. Том Выпуск 17. — Нижневартовск: Нижневартовский государственный университет, 2020. — С. 174–178. — EDN DOGVCH.
6. Летон Г. Исследование методологии оптимизации рендеринга компонентов на примере Svelte / Г. Летон, П.Е. Глазько // XI Конгресс молодых учёных: Сборник научных трудов, Санкт-Петербург, 04–08 апреля 2022 года. — Санкт-Петербург: федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО», 2022. — С. 231–234. — EDN TJCBNX.
7. Мельников М.О. Оптимизация Django веб-приложений с помощью пулера соединений pgbouncer / М.О. Мельников // Системы управления, сложные системы: моделирование, устойчивость, стабилизация, интеллектуальные технологии: материалы VII Международной научно-практической конференции, Елец, 22–23 апреля 2021 года. — Елец: Елецкий государственный университет им. И.А. Бунина, 2021. — С. 211–214. — EDN GCRRGE.
8. Шенгелия А.Р. Веб-аналитика как инструмент повышения эффективности бизнеса / А.Р. Шенгелия, А.Н. Норкина // Финансовая безопасность. Современное состояние и перспективы развития: Материалы VIII Международной научно-практической конференции Международного сетевого института в сфере ПОД/ФТ, Москва, 14–15 декабря 2022 года. Том 1. — Москва: Национальный исследовательский ядерный университет «МИФИ», 2022. — С. 116–121. — EDN EWVLSA.
9. Node.js / [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Node.js (дата обращения: 12.09.2023).
10. Robert Dempsey. ES5 vs ES6+ JavaScript Performance Comparisons / [Электронный ресурс]. URL: https://medium.com/javascript-in-plain-english/es5-vs-es6-performance-comparisons-c3606a241633 (дата обращения: 12.09.2023).
11. Statcounter Global Stats. Browser Market Share Worldwide / [Электронный ресурс]. URL: https://gs.statcounter.com/browser-market-share (дата обращения: 12.09.2023).

Журнал «Современная наука: актуальные проблемы теории и практики»