12 советов по улучшению производительности JavaScript-приложений

[Автор оригинала: Liz Parody](https://nodesource.com/blog/improve-javascript-performance)

* [Блог компании RUVDS.com](https://habr.com/ru/company/ruvds/),
* [Разработка веб-сайтов](https://habr.com/ru/hub/webdev/),
* [JavaScript](https://habr.com/ru/hub/javascript/)
* Перевод

Производительность — это один из важнейших вопросов, встающих перед разработчиками веб-страниц или веб-приложений. Никто не будет рад «падающему» от чрезмерной нагрузки приложению, или странице, которая загружается целую вечность. Пользователи веб-сайтов не готовы слишком долго ждать их загрузки или приведения их страниц в рабочее состояние. Согласно данным [Kissmetrics](https://www.nngroup.com/articles/how-long-do-users-stay-on-web-pages/), 47% посетителей ожидают, что веб-сайт загрузится менее чем за 2 секунды. 40% посетителей покинут сайт в том случае, если его загрузка займёт более 3 секунд.  
  
  
  
Автор материала, перевод которого мы сегодня публикуем, говорит, что, если учитывать вышеприведённые цифры, становится ясно, что производительность — это то, о чём всегда стоит помнить веб-разработчикам. Здесь будут приведены 12 рекомендаций по улучшению производительности JS-проектов.

1. Пользуйтесь кэширующими механизмами браузеров

Существует два основных способа кэширования данных средствами браузеров. Первый — использование JavaScript-API Cache, работа с которым осуществляется с помощью сервис-воркеров. Второй — это обычный HTTP-кэш.  
  
Скрипты часто используются для организации доступа к определённым объектам. Если хранить ссылку на объект, к которому часто требуется доступ, в переменной, а также если использовать эту переменную в повторяющихся операциях, при выполнении которых требуется доступ к объекту, можно добиться улучшения производительности кода.

2. Оптимизируйте код в расчёте на те среды, в которых он будет выполняться

Для того чтобы адекватно оценивать улучшения, вносимые в программу, рекомендуется сформировать набор сред, в которых можно провести измерения.  
  
На практике вам не удастся выполнять исследования производительности кода, например, во всех существующих версиях JS-движков, равно как и оптимизировать код в расчёте на все среды, в которых он может выполняться. Но надо отметить, что тестирование кода в какой-то одной среде — тоже не лучшая практика. Такой подход может дать искажённые результаты. Поэтому важно сформировать набор сред, в которых, вероятнее всего, будет выполняться код, и тестировать проекты в этих средах.

3. Избавляйтесь от неиспользуемого JS-кода

Благодаря удалению из проекта неиспользуемого кода улучшится не только время загрузки скриптов браузерами, но и время, необходимое браузерам на то, чтобы проанализировать и скомпилировать код. Для того чтобы избавиться от неиспользуемого кода стоит обратить внимание на особенности работы проекта. Так, если вы обнаружили некий функционал, с которым не работают пользователи — рассмотрите возможность убрать его из проекта, а заодно — и связанный с ним JS-код. В результате сайт будет загружаться быстрее, он будет быстрее подготавливаться к работе в браузере. Это благотворно скажется на тех впечатлениях, которые работа с сайтом вызовет у пользователей. Анализируя проект, учитывайте, что, например, некая библиотека, включённая в его состав, может быть включена в него по ошибке. Она вполне может реально в нём не использоваться. От неё стоит избавиться. То же самое касается использования неких зависимостей, которые реализуют то, что уже реализовано в современных браузерах. Как результат, переход на стандартные возможности браузеров, дублируемые этой зависимостью, поможет избавиться от ненужного кода.

4. Экономно расходуйте память

Стоит стремиться к тому, чтобы веб-проекты использовали бы лишь ту память, без которой они абсолютно не в состоянии обойтись. Дело в том, что разработчику нельзя заранее узнать о том, сколько памяти может быть доступно его приложению на некоем устройстве. Если приложение неоправданно использует большие объёмы памяти — это создаёт повышенную нагрузку на механизмы управления памятью браузерного JS-движка. В частности, это касается сборщика мусора. Частые вызовы сборщика мусора приводят к замедлению работы программ. Это негативно влияет на удобство работы с проектом.

5. Используйте механизмы отложенной загрузки для второстепенных скриптов

Пользователи хотят, чтобы веб-страницы загружались бы как можно быстрее. Но вряд ли для начального отображения страницы нужен абсолютно весь JS-код проекта. Если пользователю, чтобы задействовать некий код, нужно выполнить какое-то действие (например, щёлкнуть по некоему элементу или перейти на какую-нибудь вкладку в приложении), то загрузку этого кода можно отложить, выполнив её после первоначальной загрузки страницы и самых важных ресурсов.  
  
При таком подходе можно избежать загрузки и компилирования браузером большого объёма JS-кода в самом начале работы, то есть — избежать замедления вывода страницы, вызванного необходимостью выполнения этих операций. После того, как завершится загрузка всего самого важного, можно начать загружать дополнительный код. В результате, когда этот код понадобится пользователю, он уже будет ему доступен. В соответствии с моделью [RAIL](https://developers.google.com/web/fundamentals/performance/rail) Google рекомендует выполнять сеансы отложенной загрузки скриптов длительностью порядка 50 мс. При таком подходе операции по загрузке кода не повлияют на взаимодействие пользователя со страницей.

6. Избегайте утечек памяти

Если в вашем приложении случилась утечка памяти, то это выразится в том, что загруженная страница будет запрашивать у браузера всё больше и больше памяти. В результате потребление памяти этой страницей может достичь такого уровня, что это плохо повлияет на производительность всей системы. Вероятно, вы и сами сталкивались с подобной проблемой (и вам это, наверняка, не понравилось). Вполне возможно, что страница, на которой имелась утечка памяти, содержала некое средство просмотра изображений — вроде слайдера или «карусели».  
  
В инструментах разработчика Chrome можно проанализировать сайт на предмет утечек памяти. Делается это путём исследования показателей средствами вкладки Performance. Обычно утечки памяти исходят из-за фрагментов DOM, удалённых со страницы, но привязанным к каким-то переменным. Это не позволяет сборщику мусора очистить память, занимаемую данными ненужными фрагментами DOM.

7. Если вам нужно выполнить некие тяжёлые вычисления — используйте веб-воркеры

Из материалов ресурса MDN можно узнать о том, что веб-воркеры позволяют запускать код в фоновом потоке, отделённом от главного потока веб-приложения. Преимущество такого подхода заключается в том, что тяжёлые вычисления могут быть выполнены в отдельном потоке. Это позволяет главному потоку (обычно ответственному за обеспечение работы пользовательского интерфейса) выполняться без блокировок и замедлений.  
  
Веб-воркеры позволяют выполнять вычисления, интенсивно использующие процессор, и при этом не блокировать поток пользовательского интерфейса. Эта технология позволяет создавать новые потоки и давать им задания, что благотворно сказывается на производительности приложения. При таком подходе задачи, на выполнение которых требуется много времени, не блокируют выполнение других задач. При выполнении подобных задач в главном потоке выполнение других задач блокируется.

8. Если вы обращаетесь к элементу DOM несколько раз — сохраните ссылку на него в переменной

Получение ссылки на элемент DOM — операция медленная. Если вы собираетесь обращаться к элементу несколько раз — лучше всего сохранить ссылку на него в локальной переменной. Но тут важно помнить о том, что если элемент, ссылка на который хранится в переменной, будет позже из DOM удалён, нужно убрать из переменной и ссылку на него. Например, сделать это можно, записав в переменную значение null. Это позволит избежать утечек памяти.

9. Стремитесь объявлять переменные в той же области видимости, в которой они будут использоваться

JavaScript, при попытке доступа к переменной, сначала ищет её в локальной области видимости. Если её там не оказывается — поиск продолжается в областях видимости, в которую вложена локальная область видимости. Так происходит до тех пор, пока проверке не подвергнутся глобальные переменные. Сохранение переменных в локальных областях видимости ускоряет доступ к ним.  
  
Постарайтесь, без особой необходимости, не использовать при объявлении переменных ключевое слово var. Используйте вместо него, для объявления, соответственно, переменных и констант, ключевые слова let и const. Они отличаются блочной областью видимости и некоторыми другими полезными особенностями. Внимательно относитесь к использованию переменных в функциях, стремясь к тому, чтобы переменные, к которым вы обращаетесь внутри функции, были бы для неё локальными. Помните о неприятностях, которые может вызвать неявное объявление глобальных переменных.

10. Старайтесь не использовать глобальные переменные

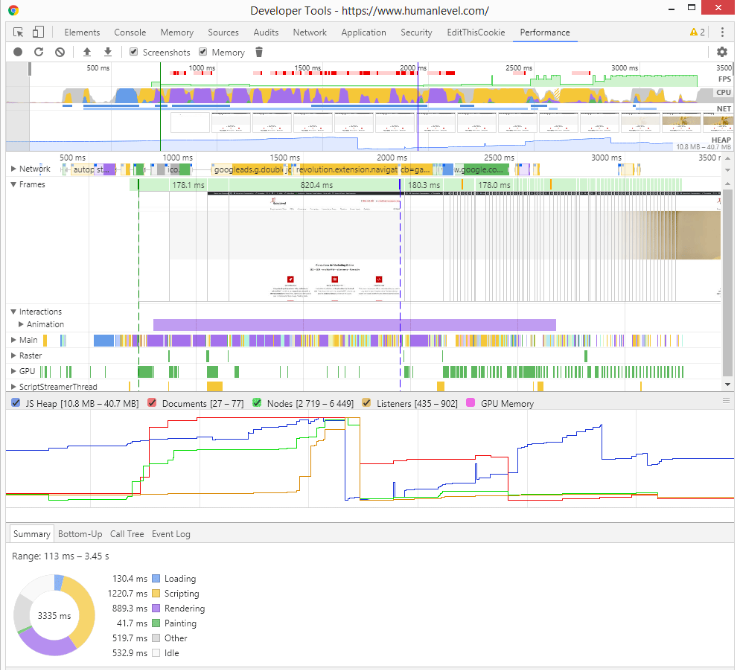
Глобальные переменные существуют в течение всего времени работы скрипта. Локальные же переменные уничтожаются при уничтожении локальной области видимости. Поэтому глобальными переменными стоит пользоваться лишь тогда, когда это действительно необходимо.

11. Применяйте в JavaScript оптимизации кода, которые вы применяли бы и к программам, написанным на других языках

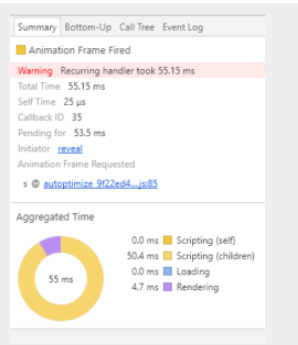
* Всегда применяйте алгоритмы с наименьшей из возможных вычислительной сложностью, решайте задачи с использованием оптимальных структур данных.
* Оптимизируйте используемые алгоритмы, стремясь получить те же результаты, выполнив меньше вычислений.
* Избегайте рекурсивных вызовов.
* Оформляйте повторяющиеся фрагменты вычислений в виде функций.
* Упрощайте математические вычисления.
* Используйте поисковые массивы вместо конструкций switch/case.
* Стремитесь к тому, чтобы условия, проверяемые в условных конструкциях, чаще принимали бы истинные значения. Это способствует более эффективному использованию возможностей процессора по упреждающему исполнению команд.
* Если у вас есть возможность использовать для выполнения неких действий побитовые операторы — сделайте это. На выполнение подобных вычислений уходит меньше ресурсов процессора.

12. Используйте инструменты для исследования производительности приложений

Для исследования различных аспектов веб-проектов можно порекомендовать инструмент Lighthouse. Он выставляет приложению оценки по следующим показателям: Performance, Progressive Web App, Accessibility, Best Practices, SEO. Lighthouse не только выставляет оценки, но и даёт рекомендации по улучшению проекта. Ещё одно средство для анализа производительности, [Google PageSpeed](https://developers.google.com/speed/pagespeed/insights/?hl=en), создано для того, чтобы помочь разработчикам исследовать свои сайты и увидеть направления их возможного улучшения.  
  
И Lighthouse, и PageSpeed — инструменты не идеальные, но их использование помогает увидеть проблемы, которые, на первый взгляд, могут оказаться незаметными.  
  
В меню Chrome можно найти команду, открывающую диспетчер задач. Там выводятся сведения о системных ресурсах, используемых открытыми вкладками браузера. Более подробные сведения о том, что происходит на странице, можно получить, открыв вкладку Performance инструментов разработчика Chrome (подобные инструменты есть и в других браузерах). Эта вкладка позволяет анализировать множество показателей, касающихся производительности сайта.



*Вкладка Performance в инструментах разработчика Chrome*  
  
В ходе сбора сведений о производительности страниц средствами Chrome можно настраивать доступные страницам процессорные и сетевые ресурсы, что позволяет выявлять и исправлять проблемы.



*Анализ производительности страницы в Chrome*  
  
Для того чтобы глубже проанализировать веб-сайт — можно воспользоваться API Navigation Timing. Оно позволяет выполнять измерения различных показателей прямо в коде приложения.  
  
Если вы разрабатываете на JavaScript серверные проекты с использованием Node.js, то вам, для глубокого анализа своих приложений, можно воспользоваться платформой NodeSource. Измерения, проводимые средствами этой платформы, оказывают незначительное воздействие на проект. В среде Node.js, как и в браузере, может возникать множество проблем — вроде тех же утечек памяти. Анализ проектов, основанных на Node.js, помогает выявлять и устранять проблемы с их производительностью.

Итоги

Важно поддерживать баланс между оптимизацией кода и его читабельностью. Код интерпретируется компьютером, но поддерживать его приходится людям. Поэтому код должен быть понятным не только компьютеру, но и человеку.  
  
Кроме того, полезно помнить о том, что производительность стоит принимать в расчёт всегда, но она не должна быть важнее обеспечения безошибочной работы кода и реализации нужных пользователям возможностей приложений.