

РАЗРАБОТКА ВЕБ-САЙТОВ*, ПОИСКОВАЯ ОПТИМИЗАЦИЯ, JAVASCRIPT*, БЛОГ
КОМПАНИИ RUVDS.COM

Руководство по SEO JavaScript-сайтов. Часть 1. Интернет глазами Google

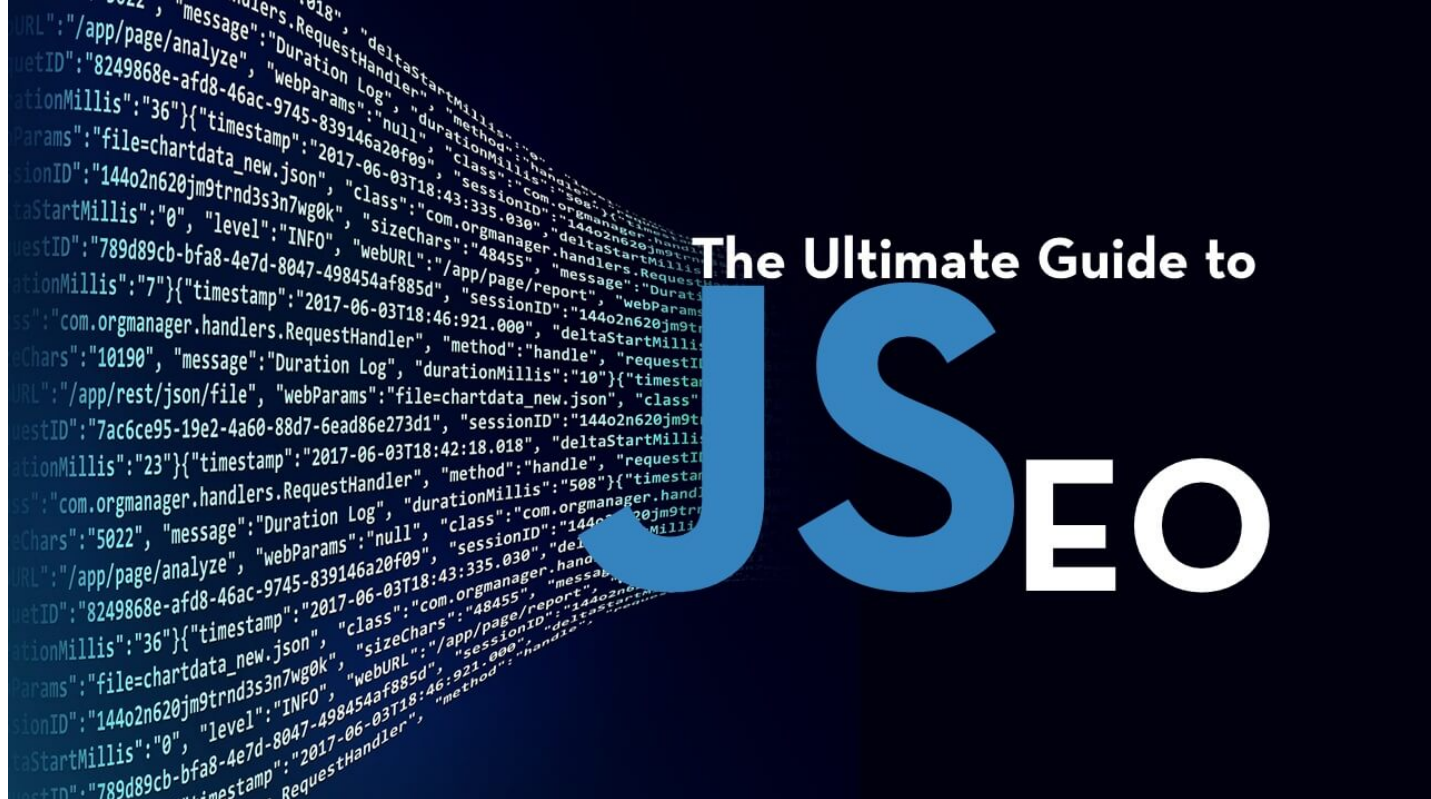
ПЕРЕВОД

ru_vds 12 марта в 14:48  7,1k

Оригинал: [Tomasz Rudzki](#)

Представляем вашему вниманию перевод первой части материала, который посвящён поисковой оптимизации сайтов, построенных с использованием JavaScript. Речь пойдёт об особенностях сканирования, анализа и индексирования таких сайтов поисковыми роботами, о проблемах, сопутствующих этим процессам, и о подходах к решению этих проблем.

В частности, сегодня автор этого материала, Томаш Рудски из компании Elephate, расскажет о том, как сайты, которые используют современные JS-фреймворки, вроде Angular, React, Vue.js и Polymer, выглядят с точки зрения Google. А именно, речь пойдёт о том, как Google обрабатывает сайты, о технологиях, применяемых для анализа страниц, о том, как разработчик может проанализировать сайт для того, чтобы понять, сможет ли Google нормально этот сайт проиндексировать.



The Ultimate Guide to

JS SEO

JavaScript-технологии разработки веб-сайтов в наши дни весьма популярны, поэтому может показаться, что они уже достигли достаточно высокого уровня развития во всех мыслимых направлениях. Однако, в реальности всё не так. В частности, разработчики и SEO-специалисты всё ещё находятся в самом начале пути к тому, чтобы сделать сайты, построенные на JS-фреймворках, успешными в плане их взаимодействия с поисковыми системами. До сих пор множество подобных сайтов, несмотря на их популярность, занимают далеко не самые высокие места в поисковой выдаче Google и других поисковых систем.

Может ли Google сканировать и анализировать сайты, основанные на JavaScript?

Ещё в 2014-м компания Google [заявляла](#) о том, что их системы неплохо индексируют сайты, использующие JavaScript. Однако, несмотря на эти заявления, всегда давались рекомендации

осторожно относиться к этому вопросу. Взгляните на это извлечение из оригинала материала «[Совершенствуем понимание веб-страниц](#)» (здесь и далее выделение сделано автором материала):

«К сожалению, индексация не всегда проходит гладко, что может привести к проблемам, влияющим на позицию вашего сайта в результатах поиска ... Если код JavaScript слишком сложный или запутанный, Google может проанализировать его некорректно... Иногда JavaScript удаляет контент со страницы, а не добавляет его, что также затрудняет индексацию».



Возможность сканирования, возможность анализа и бюджет сканирования

Для того, чтобы понять, сможет ли поисковая система правильно обработать сайт, то есть, обработать его так, как того ожидает создатель сайта, нужно учесть три фактора:

1. Возможность сканирования сайта: системы Google должны быть способны просканировать сайт, учитывая его структуру.
2. Возможность анализа сайта: системы Google не должны испытывать проблем в ходе анализа сайта с использованием технологий, используемых для формирования его страниц.
3. Бюджет сканирования: времени, выделенного Google на обработку сайта, должно хватить для его полноценной индексации.

О клиентском и серверном рендеринге

Говоря о том, может ли Google сканировать и анализировать сайты, использующие JavaScript, нам нужно затронуть две очень важные концепции: рендеринг, или визуализация страниц, на стороне сервера, и на стороне клиента. Эти идеи необходимо понимать каждому специалисту по SEO, который имеет дело с JavaScript.

При традиционном подходе (серверный рендеринг), браузер или робот Googlebot загружают с сервера HTML-код, который полностью описывает страницу. Все необходимые материалы уже готовы, браузеру (или роботу) нужно лишь загрузить HTML и CSS и сформировать готовую к просмотру или анализу страницу. Обычно поисковые системы не испытывают никаких проблем с индексацией сайтов, использующих серверный рендеринг.

Всё большую популярность получает метод визуализации страниц на стороне клиента, который имеет определённые особенности. Эти особенности иногда приводят к проблемам с анализом таких

страниц поисковыми системами. Весьма распространена ситуация, когда при первоначальной загрузке данных браузер (или Googlebot) получает пустую HTML-страницу.

Фактически, на такой странице либо вовсе нет данных, подходящих для анализа и индексации, либо их очень мало. Затем в дело вступают JavaScript-механизмы, которые асинхронно загружают данные с сервера и обновляют страницу (изменяя DOM).

Если на вы используете методику визуализации на стороне клиента, вам нужно убедиться в том, что Google в состоянии правильно сканировать и обрабатывать страницы вашего сайта.

JavaScript и ошибки

HTML и JS коренным образом различаются в подходах к обработке ошибок. Единственная ошибка в JavaScript-коде может привести к тому, что Google не сможет проиндексировать страницу.

Позвольте мне процитировать Матиаса Шафера, автора работы [«Надёжный JavaScript»](#): «JS-парсер не отличается дружелюбием. Он совершенно нетерпим к ошибкам. Если он встречает символ, появление которого в определённом месте не ожидается, он немедленно прекращает разбор текущего скрипта и выдаёт `SyntaxError`. Поэтому единственный символ, находящейся не там, где нужно, единственная опечатка, может привести к полной неработоспособности скрипта.»

Ошибки разработчиков фреймворков

Возможно вы слышали об [эксперименте](#) по SEO в применении к JavaScript-сайтам, который провёл Бартош Горалевич, занимающий должность CEO в компании Elephate, для того, чтобы узнать, может ли Google индексировать веб-сайты, созданные с использованием распространённых JS-фреймворков.

Yo!

Below is a list of simple Hello World projects utilising different JS frameworks used for JS crawling tests.



This page is an SEO experiment designed by [Kamil Grymuza \(JS development\)](#) & [Bartosz Goralewicz \(JS SEO\)](#).

Список исследованных технологий

В самом начале выяснилось, что Googlebot не в состоянии анализировать страницы, созданные с использованием Angular 2.

Это было странно, так как Angular создан командой Google, поэтому Бартош решил выяснить причины происходящего. [Вот что он пишет по этому поводу:](#)

«Оказалось, что имелась ошибка в QuickStart Angular 2, в чём-то вроде учебного руководства, посвящённого тому, как готовить к работе проекты, основанные на этом фреймворке. Ссылка на это руководство присутствовала в официальной документации. Было выяснено, что команда Google Angular допустила ошибку, которая была исправлена 26-го апреля 2017 года».



The screenshot shows a GitHub commit titled "Fix systemjs-angular-loader.js for <=IE10 (#443)" by nielsheeren. The commit message states: "let is ES2015 and therefore not supported by <=IE10". The commit is dated April 26, 2017. Below the commit information, a diff for the file `src/systemjs-angular-loader.js` is shown. The diff highlights a change on line 22, where the code was changed from `let resolvedUrl = url;` to `var resolvedUrl = url;`. The diff interface includes buttons for "Unified", "Split", and "View".

```
Fix systemjs-angular-loader.js for <=IE10 (#443)
"let" is ES2015 and therefore not supported by <=IE10

master (#443)
nielsheeren committed with Foxandxss on Apr 26
1 parent 16232d7 | commit 418cc2e1d00bf8a5145e761b7c92d8bb4440046c

Showing 1 changed file with 1 addition and 1 deletion.
Unified Split

2 src/systemjs-angular-loader.js View
@@ -19,7 +19,7 @@ module.exports.translate = function(load){
 19 19
 20 20     load.source = load.source
 21 21     .replace(templateUrlRegex, function(match, quote, url){
22 22 -     let resolvedUrl = url;
22 22 +     var resolvedUrl = url;
 23 23
 24 24     if (url.startsWith('.')) {
 25 25         resolvedUrl = basePath + url.substr(1);
 26 26
 27 27     }
```

Ошибка в Angular 2

Исправление ошибки привело к возможности нормального индексирования тестового сайта на Angular 2.

Этот пример отлично иллюстрирует ситуацию, когда единственная ошибка может привести к тому, что Googlebot оказывается не в состоянии проанализировать страницу.

Масла в огонь подливает и то, что ошибка была совершена не новичком, а опытным человеком, участвующим в разработке Angular, второго по популярности JS-фреймворка.

Вот ещё один отличный пример, который, по иронии судьбы, снова связан с Angular. В декабре 2017-го Google [исключила из индекса](#) несколько страниц сайта Angular.io (веб-сайт, основанный на Angular 2+, на котором применяется технология визуализации на стороне клиента). Почему это произошло? Как вы можете догадываться, одна ошибка в их коде сделала невозможной визуализацию страниц средствами Google и привела к масштабному исключению страниц из индекса. Позже ошибка была исправлена.

Вот как Игорь Минар из Angular.io это [объяснил](#):

«Учитывая то, что мы не меняли проблематичный код в течение 8 месяцев, и мы столкнулись со значительным падением трафика с поисковых систем, начиная примерно с 11 декабря 2017, я полагаю, что за это время что-то изменилось в системе сканирования сайтов, что и привело к тому, что большая часть сайта оказалась исключённой из поискового индекса, что, в свою очередь, стало причиной падения посещаемости ресурса.»

Исправление вышеупомянутой ошибки, препятствующей анализу страниц ресурса Angular.io, было возможным благодаря опытной команде JS-разработчиков, и тому факту, что они реализовали

ведение журнала ошибок. После того, как ошибка была исправлена, Google снова смог проиндексировать проблемные страницы.

О сложности сканирования сайтов, построенных с использованием JavaScript

Вот как происходит сканирование и индексирование обычных HTML-страниц. Тут всё просто и понятно:

1. Googlebot загружает HTML-файл.
2. Googlebot извлекает ссылки из кода страницы, в результате он может параллельно обрабатывать несколько страниц.
3. Googlebot загружает CSS-файлы.
4. Googlebot отправляет все загруженные ресурсы системе индексирования (Caffeine).
5. Caffeine индексирует страницу.

Всё это происходит очень быстро.

Однако процесс усложняется, если работа ведётся с веб-сайтом, основанным на JavaScript:

1. Googlebot загружает HTML-файл.
2. Googlebot загружает CSS и JS-файлы.
3. После этого Googlebot должен использовать Google Web Rendering Service (WRS) (эта система является частью Caffeine) для того, чтобы разобрать, скомпилировать и выполнить JS-код.

4. Затем WRS получает данные из внешних API, из баз данных, и так далее.
5. После того, как страница собрана, её, в итоге, может обработать система индексирования.
6. Только теперь робот может обнаружить новые ссылки и добавить их в очередь сканирования.

Весь этот процесс гораздо сложнее, чем сканирование HTML-сайтов. Во внимание тут нужно принять следующее:

- Разбор, компиляция и выполнение JS — это операции, которые требуют немалых затрат времени.
- В случае с сайтами, на которых интенсивно используется JavaScript, Google приходится ждать до тех пор, пока будут выполнены все вышеописанные шаги прежде чем можно будет проиндексировать содержимое страниц.
- Процесс сборки страницы — это не единственная медленная операция. Это также относится к процессу обнаружения новых ссылок. На сайтах, интенсивно использующих JS, Google обычно не может обнаружить новые ссылки до того, как страница не будет полностью сформирована.

Индексирование HTML-страницы и страницы, формируемой средствами JS

Теперь мне хотелось бы проиллюстрировать проблему сложности JavaScript-кода. Готов поспорить, что 20-50% посетителей вашего сайта просматривают его с мобильного устройства. Знаете ли вы о том, сколько времени занимает разбор 1 Мб JS-кода на мобильном устройстве? По информации [Сэма Сакконе из Google](#), Samsung Galaxy S7 тратит на это примерно 850 мс, а Nexus 5 — примерно 1700 мс! После разбора JS-кода его ещё нужно скомпилировать и выполнить. А на счету — каждая секунда.

Если вы хотите больше узнать о бюджете сканирования, советую почитать материал Барри Адамса «[JavaScript and SEO: The Difference Between Crawling and Indexing](#)». SEO-специалистам, имеющим дело с JavaScript, особенно полезными будут разделы «JavaScript = Inefficiency» и «Good SEO is Efficiency». Заодно

можете посмотреть и [этот](#) материал.

Google и браузер, выпущенный 3 года назад

Для того чтобы понять, почему при сканировании сайтов, использующих JS, у Google могут возникнуть проблемы, стоит поговорить о технических ограничениях Google.

Я уверен, что вы используете самую свежую версию вашего любимого браузера. Однако в Google дело обстоит не так. Тут, для рендеринга веб-сайтов, используется Chrome 41. Этот браузер был выпущен в марте 2015-го года. Ему уже три года! И браузер Google Chrome, и JavaScript за эти годы чрезвычайно сильно развились.

В результате оказывается, что существует множество современных возможностей, которые просто недоступны для робота Googlebot. Вот некоторые из его основных ограничений:

- Chrome 41 лишь частично поддерживает современный синтаксис JavaScript ES6. Например, он не понимает новые языковые конструкции.
- Интерфейсы, вроде IndexedDB и WebSQL, отключены.
- Куки, локальные хранилища и сессионные хранилища очищаются при перезагрузке страницы.
- И, опять же, перед нами браузер, который был выпущен три года назад!

Рассматривая технические ограничения Chrome 41, вы можете

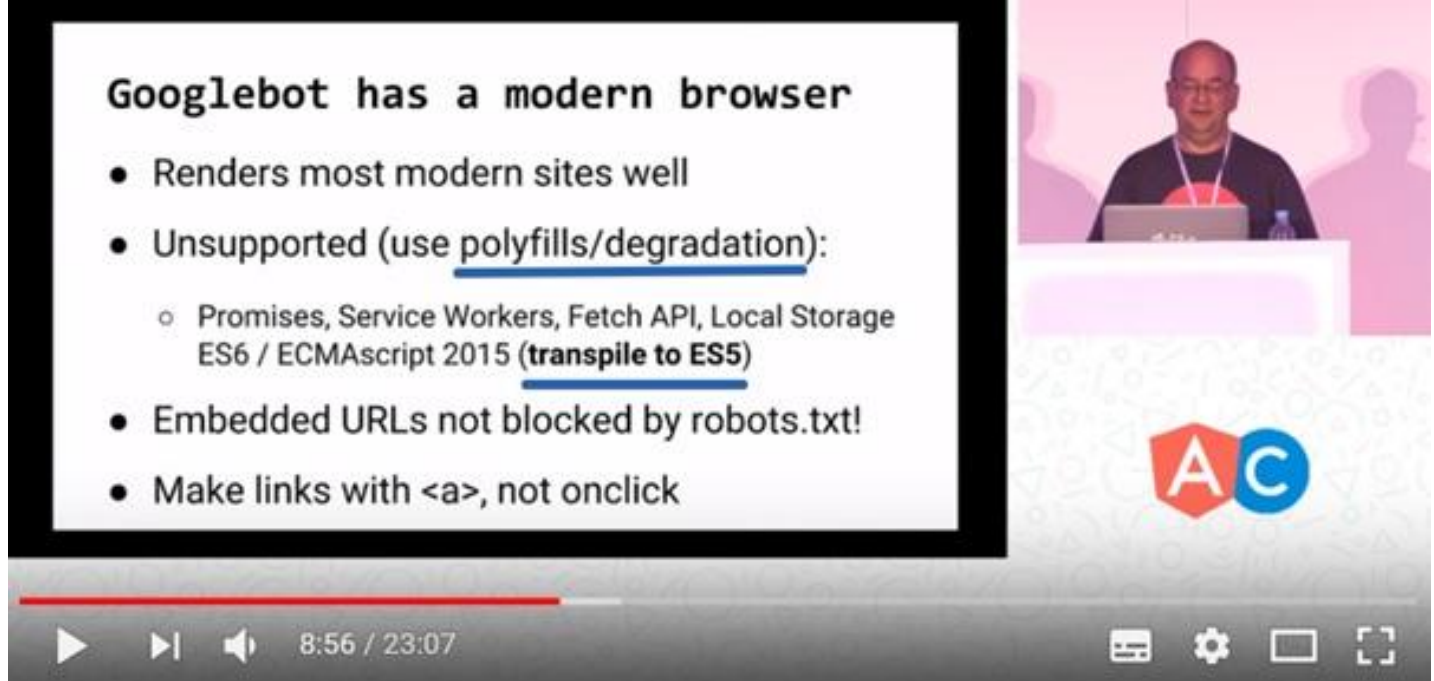
[проанализировать](#) разницу между Chrome 41 и Chrome 66 (самой свежей версией Chrome на момент написания этого материала).

Теперь, когда вы знаете, что для формирования страниц Google использует Chrome 41, найдите время на то, чтобы загрузить этот браузер и проверить собственные веб-сайты для того, чтобы понять, может ли этот браузер нормально с ними работать. Если нет — загляните в консоль Chrome 41 для того, чтобы попытаться узнать, что может быть причиной ошибок.

Кстати, раз уж заговорили об этом, [вот мой материал](#), посвящённый работе с Chrome 41.

Современные возможности JavaScript и индексирование сайтов

Как быть тому, кто стремится использовать современные возможности JS, но при этом, хочет, чтобы Google нормально индексировал его сайты? На этот вопрос отвечает данный кадр из видео:



Googlebot has a modern browser

- Renders most modern sites well
- Unsupported (use polyfills/degradation):
 - Promises, Service Workers, Fetch API, Local Storage ES6 / ECMAScript 2015 (transpile to ES5)
- Embedded URLs not blocked by robots.txt!
- Make links with <a>, not onclick

8:56 / 23:07

Современные возможности JavaScript и индексирование страниц

Браузер, применяемый в Google для формирования страниц сайтов, основанных на JS, может правильно обрабатывать сайты, использующие современные возможности JS, однако, разработчикам таких сайтов потребуется приложить для этого некоторые усилия. А именно, использовать полифиллы, создавать упрощённую версию сайта (используя технику постепенной деградации) и выполнять транспилиацию кода в ES5.

Постепенная деградация и полифиллы

Популярность JavaScript росла очень быстро, а теперь это происходит быстрее, чем когда бы то ни было. Однако некоторые возможности JavaScript просто не реализованы в устаревших браузерах (так уж совпало, что Chrome 41 — это как раз такой браузер). Как результат, нормальный рендеринг сайтов, использующих современные возможности JS, в таких браузерах

невозможен. Однако, веб-мастера могут это обойти, используя технику постепенной деградации.

Если вам хочется реализовать некоторые современные возможности JS, которые поддерживают лишь немногие браузеры, в таком случае вам нужно обеспечить использование упрощённой версии вашего веб-сайта в других браузерах. Помните о том, что версию Chrome, которую использует Googlebot, определённо, нельзя считать современной. Этому браузеру уже три года.

Выполняя анализ браузера, вы можете в любое время проверить, поддерживает он некую возможность или нет. Если он эту возможность не поддерживает, вы, вместо неё, должны предложить нечто такое, что подходит для этого браузера. Эта замена и называется полифиллом.

Кроме того, если нужно, чтобы сайт мог быть обработан поисковыми роботами Google, вам совершенно необходимо использовать транспиляцию JS-кода в ES5, то есть, преобразование тех конструкций JS, которые не понимает Googlebot, в конструкции, понятные ему.

Например, когда транспилятор встречает выражение `let x=5` (большинство старых браузеров эту конструкцию не поймут), он преобразует его в выражение `var x=5` (эта конструкция понятна всем браузерам, в том числе и Chrome 41, который играет для нас особую роль).

Если вы используете современные возможности JavaScript и

хотите, чтобы ваши сайты правильно обрабатывались поисковыми роботами Google, вам, определённо, стоит использовать транспиляцию в ES5 и полифиллы.

Тут я стараюсь объяснить всё это так, чтобы было понятно не только JS-разработчикам, но и, например, SEO-специалистам, которые далеки от JavaScript. Однако, в детали мы не вдаёмся, поэтому, если вы чувствуете потребность лучше разобраться в том, что такое полифиллы — взгляните на [этот материал](#).

Googlebot — это не настоящий браузер

Когда вы путешествуете по интернету, ваш браузер (Chrome, Firefox, Opera, или любой другой) загружает ресурсы (изображения, скрипты, стили) и показывает вам страницы, из всего этого собранные.

Однако, Googlebot работает не так, как обычный браузер. Его цель — проиндексировать всё, до чего он может дотянуться, загружая при этом только самое важное.

Всемирная паутина — это огромное информационное пространство, поэтому Google оптимизирует систему сканирования с учётом производительности. Именно поэтому Googlebot иногда не посещает все страницы, на посещение которых рассчитывает веб-разработчик.

Ещё важнее то, что алгоритмы Google пытаются выявить ресурсы, которые необходимы с точки зрения формирования страницы.

Если какой-то ресурс выглядит, с этих позиций, не особенно важным, он попросту может быть не загружен роботом Googlebot.

Googlebot and WRS prioritize essential page content

Googlebot is designed to be a good citizen of the web. Crawling is its **main priority**, while making sure it doesn't degrade the experience of users visiting the site. Googlebot and WRS continuously analyze and identify resources that don't contribute to essential page content and may not fetch such resources. For example, reporting and error requests that don't contribute to essential page content, and other similar types of requests are unused or unnecessary to extract essential page content.

★ **Note:** Client-side analytics may not provide a full or accurate representation of Googlebot and WRS activity on your site. Use [Search Console](#) to monitor Googlebot and WRS activity and feedback on your site.

Googlebot и WRS избирательно загружают материалы, выбирая только самые важные

В результате может оказаться так, что сканер не станет загружать некоторые из ваших JS-файлов, так как его алгоритмы решили, что, с точки зрения формирования страницы, они не важны. То же самое может произойти и из-за проблем с производительностью (то есть, в ситуации, когда выполнение скрипта занимает слишком много времени).

Хочу отметить, что Том Энтони заметил одну **интересную особенность в поведении робота Googlebot**. Когда используется JS-функция `setTimeout`, настоящий браузер получает указание подождать определённое время. Однако, Googlebot не ждёт, он выполняет всё немедленно. Этому не стоит удивляться, так как цель роботов Google заключается в том, чтобы проиндексировать весь интернет, поэтому их оптимизируют с учётом производительности.

Правило пяти секунд

Многие эксперименты в области SEO указывают на то, что, в целом, Google не может ждать окончания выполнения скрипта, который выполняется более 5 секунд. Мои эксперименты, похоже, это подтверждают.

Не принимайте это как должное: добейтесь того, чтобы ваши JS-файлы загружались и завершали работу менее чем за 5 секунд.

Если ваш веб-сайт загружается по-настоящему медленно, вы можете многое потерять. А именно:

- Посетителям сайта будет некомфортно с ним работать, они могут его покинуть.
- У Google могут возникнуть проблемы с анализом страниц.
- Это может замедлить процесс сканирования сайта. Если страницы медленны, Google может решить, что его роботы замедляют ваш сайт и снизить частоту сканирования.

Подробнее об этом можно почитать [здесь](#).

Постарайтесь сделать ваш сайт не слишком тяжёлым, обеспечьте высокую скорость реакции сервера, а так же убедитесь в том, что сервер нормально работает под высокой нагрузкой (используйте для этого, например, [Load Impact](#)). Не усложняйте жизнь роботам Google.

Обычная ошибка в плане производительности, которую совершают разработчики, заключается в том, что они помещают

код всех компонентов страницы в единственный файл. Но если пользователь переходит на домашнюю страницу проекта, ему совершенно не нужно загружать то, что относится к разделу, предназначенному для администратора сайта. То же самое справедливо и в применении к поисковым роботам.

Для решения проблем с производительностью рекомендуется найти соответствующее руководство по применяемому JS-фреймворку. Его стоит изучить и выяснить, что можно сделать для ускорения сайта. Кроме того, советую почитать [этот](#) материал.

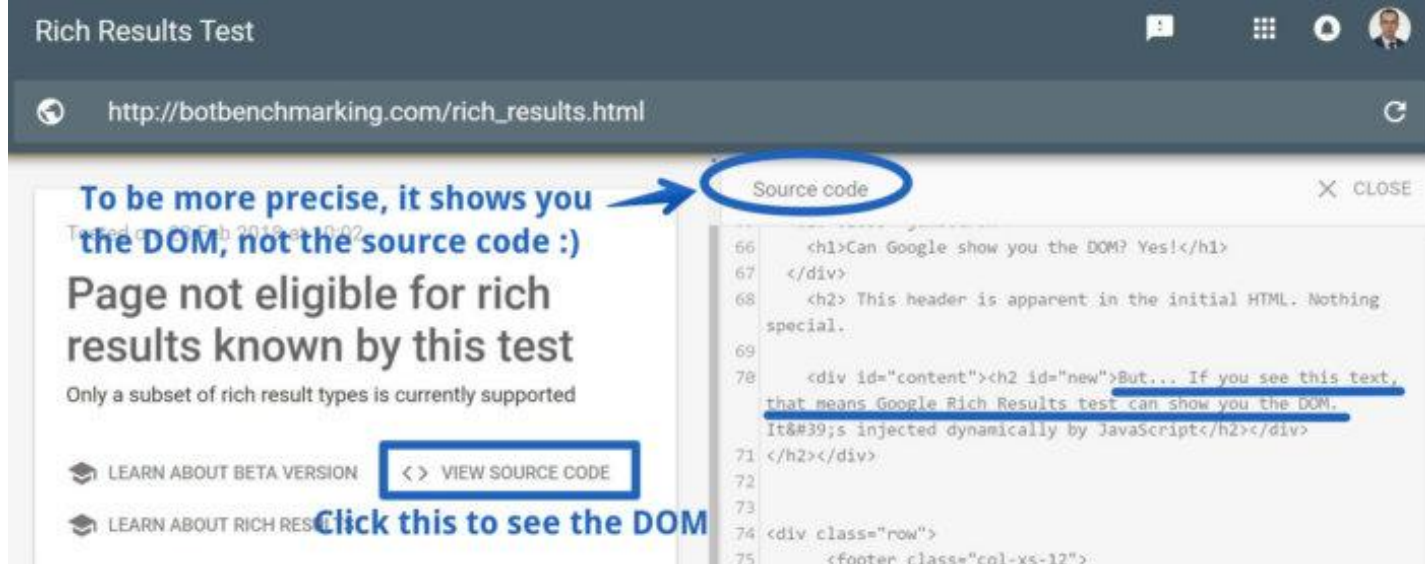
Как взглянуть на интернет глазами Google?

Если вы хотите взглянуть на интернет, и, особенно, на свой веб-сайт, глазами роботов Google, вы можете воспользоваться одним из двух подходов:

1. Используйте инструменты Fetch (сканирование) и Render (отображение) из Google Search Console (очевидно!). Но не полагайтесь на них на 100%. Настоящий Googlebot может иметь таймауты, отличающиеся от тех, которые предусмотрены в Fetch и Render.
2. Используйте Chrome 41. Как уже было сказано, достоверно известно, что Google использует для рендеринга загружаемых роботами страниц Chrome 41. Загрузить этот браузер можно, например, [здесь](#). Использование Chrome 41 имеет множество преимуществ перед применением загрузки страниц с использованием Google Search Console:

- Благодаря использованию Chrome 41 вы можете видеть журнал ошибок, выводимый в консоль браузера. Если вы столкнётесь с ошибками в этом браузере, вы можете быть практически полностью уверены в том, что и Googlebot столкнётся с теми же ошибками.
- Инструменты Fetch и Render не покажут вам результаты отрисовки DOM, а браузер покажет. Используя Chrome 41 вы можете проверить, увидит ли Googlebot ваши ссылки, содержимое панелей, и так далее.

Вот ещё один вариант: инструмент [Rich Results Test](#) (проверка расширенных результатов). Я не шучу — это средство может показать вам, как Google интерпретирует вашу страницу, демонстрируя результаты визуализации DOM, а это очень полезно. Google планирует добавить результаты рендеринга DOM в инструменты Fetch и Render. Так как до сих пор это не сделано, Джон Мюллер [советует](#) применять для этого Rich Results. Сейчас не вполне понятно, следует ли Rich Results тем же правилам рендеринга, что и система индексирования Google. Пожалуй, для того, чтобы это выяснить, стоит провести дополнительные эксперименты.



Инструмент Rich Results Test

Инструменты Fetch и Render и проверка тайм-аутов системы индексирования

Инструменты Fetch и Render могут лишь сообщить вам о том, имеет ли Google техническую возможность сформировать анализируемую страницу. Однако не полагайтесь на них, когда речь заходит о тайм-аутах. Я часто сталкивался с ситуацией, когда Fetch и Render были способны вывести страницу, но система индексирования Google не могла проиндексировать эту страницу из-за используемых этой системой тайм-аутов. Вот доказательства этого утверждения.

Я провёл [простой эксперимент](#). Первый JS-файл, включённый в анализируемую страницу, загружался с задержкой в 120 секунд. При этом технической возможности избежать этой задержки не было. Nginx-сервер был настроен на двухминутное ожидание перед выдачей этого файла.

Elements

Console

Sources

Network

>>

View:

Group by frame

Preserve log

☒

Disa

Name

bulgaria120.html

/timeouts

bootstrap.min.css

maxcdn.bootstrapcdn.com/...

jquery.min.js

ajax.googleapis.com/ajax/li...

bootstrap.min.js

maxcdn.bootstrapcdn.com/...

wait120.js

/timeouts/custom

bulgaria.js

/timeouts

×

Headers

Preview

Response

Timing

Queued at 18.00 ms

Started at 18.50 ms

Resource Scheduling

Queueing

Connection Start

Stalled

Request/Response

Request sent

Waiting (TTFB)

Content Download

[Explanation](#)

TIME

0.50 ms

TIME

1.00 ms

TIME

0

2.0 min

2.50 ms

2.0 min

Проект, используемый для эксперимента

Visiting Bulgaria - travel destination for those who think outside the box

How long Google is going to wait for necessary styles and scripts?

Script 1 is not loaded yet. Please wait...

Script 2 is not loaded yet. A few paragraphs of text will appear shortly

Check out
Fibonacci
tests

Test 20

Test 25

Test 30

Test 35

Test 40

Test 41

Test 42

Test 43

Test 44

Test 45

Test 50

Test 60

Test 70

Test 80

Test 90

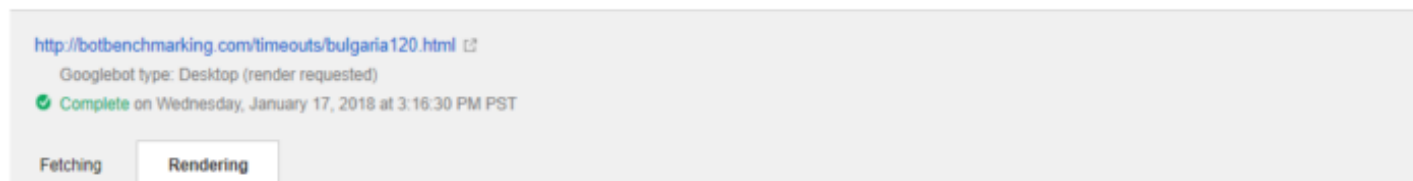
Test 100 - hidden

Test 100

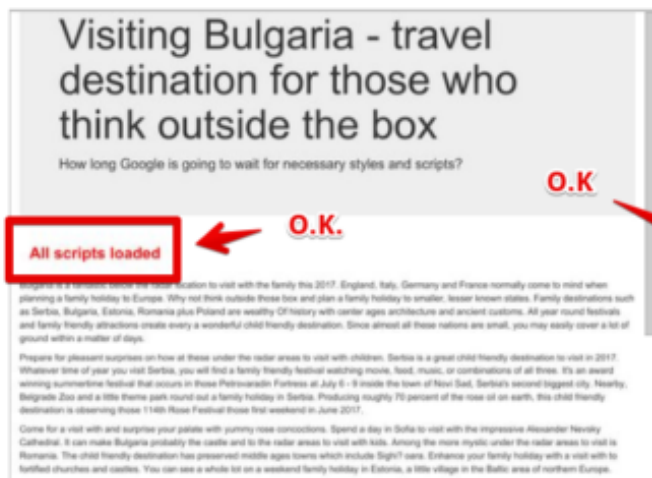
Страница экспериментального проекта

Оказалось, что инструменты Fetch и Render ждали загрузки скрипта 120 секунд (!), после чего страница выводилась правильно.

Fetch as Google



This is how Googlebot saw the page:



This is how a visitor to your website would have seen the page:



Анализ страницы средствами Fetch и Render

Однако система индексирования оказалась не такой терпеливой.



site:botbenchmarking.com/timeouts/bulgaria120.html is not loaded



All

Videos

Images

News

Maps

More

Settings

Tools

1 result (0.26 seconds)

Googlebot's patience - Googlebot benchmark

botbenchmarking.com/timeouts/bulgaria120.html ▼

Script 1 is not loaded yet. Please wait... Bulgaria is a fantastic below the radar location to visit with the family this 2017. England, Italy, Germany and France normally come to mind when planning a family holiday to Europe. Why not think outside those box and plan a family holiday to smaller, lesser known states. Family ...

Результаты индексирования экспериментального сайта

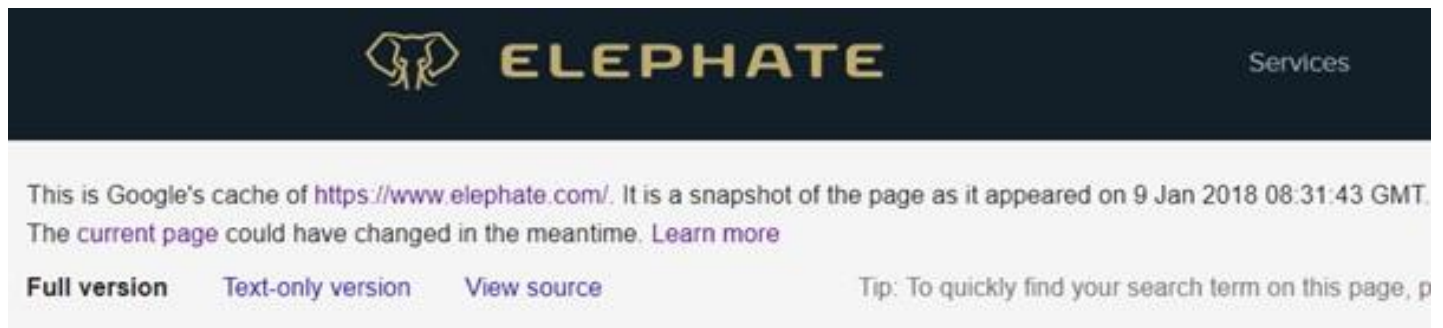
Здесь видно, что система индексирования Google просто не стала дожидаться загрузки первого скрипта и проанализировала страницу без учёта результатов работы этого скрипта.

Как видите, Google Search Console — это отличный инструмент. Однако пользоваться им следует только для проверки технической возможности анализа страницы роботами Google. Не используйте это средство для того, чтобы проверить, дожждётся ли система индексирования загрузки ваших скриптов.

Анализ кэша Google и сайты, интенсивно использующие JavaScript

Многие специалисты в области SEO применяли кэш Google для поиска проблем с анализом страниц. Однако эта методика не подходит для сайтов, интенсивно использующих JS, так как сам по себе кэш Google представляет собой исходный HTML, который Googlebot загружает с сервера (обратите внимание — это

многократно подтверждено Джоном Мюллером из Google).



Материал из кэша Google

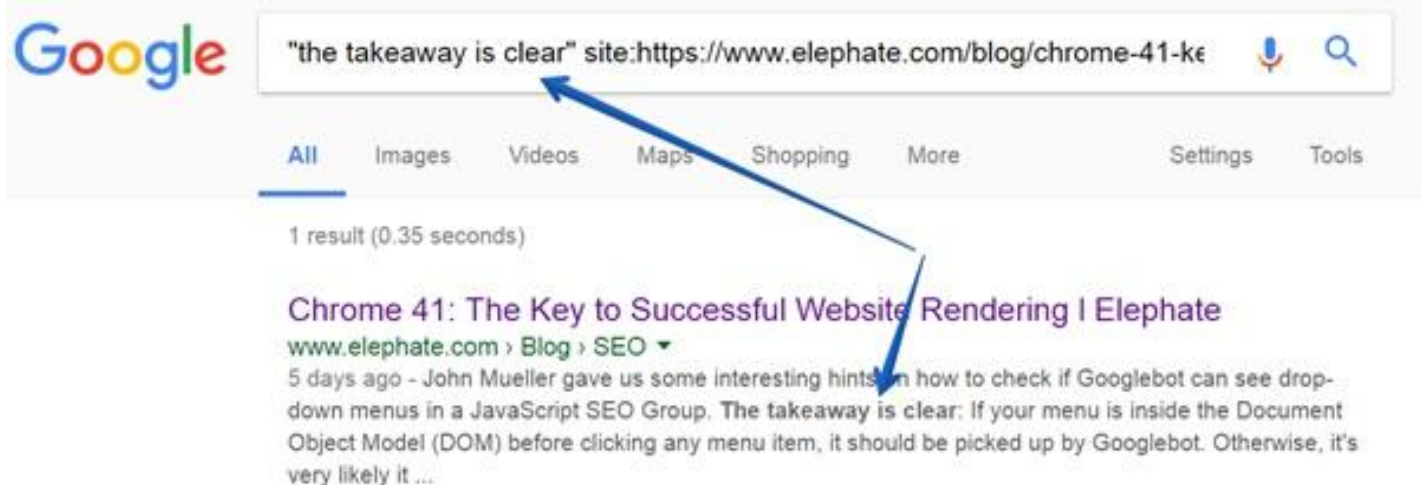
Просматривая содержимое кэша, вы видите, как ваш браузер интерпретирует HTML, собранный средствами Googlebot. Это не имеет никакого отношения к формированию страницы для целей индексирования. Если вы хотите узнать подробности о кэше Google — взгляните на [этот материал](#).

Использование команды site вместо анализа кэша Google

В настоящий момент один из лучших способов проверки того, были ли какие-то данные проиндексированы Google, заключается в использовании команды `site`.

Для того чтобы это сделать, просто скопируйте какой-нибудь фрагмент текста с вашей страницы и введите в поисковике Google команду следующего вида:

```
site:{your website} "{fragment}"
```

Команда `site`

Если вы, в ответ на такую команду, увидите в поисковой выдаче искомый фрагмент, это значит, что данные были проиндексированы.

Тут хотелось бы отметить, что подобные поисковые запросы рекомендуется выполнять в анонимном режиме. Несколько раз у меня были случаи, когда редакторы сайта меняли тексты, и по какой-то причине команда `site` сообщала о том, что проиндексированы были старые тексты. После переключения в анонимный режим браузера эта команда стала выдавать правильный результат.

Просмотр HTML-кода страницы и аудит сайтов, основанных на JS

HTML-файл представляет собой исходную информацию, которая используется браузером для формирования страницы. Я предполагаю, что вам известно, что такое HTML-документ. Он содержит сведения о разметке страницы — например, о разбиении

текста на абзацы, об изображениях, ссылках, он включает в себя команды для загрузки JS и CSS-файлов.

Посмотреть HTML-код страницы в браузере Google Chrome можно, вызвав щелчком правой кнопки мыши контекстное меню и выбрав команду View page source (просмотр кода страницы).

Однако, воспользовавшись этим режимом просмотра кода страницы, вы не увидите динамического содержимого (то есть тех изменений, которые внесены в страницу средствами JavaScript).

Вместо этого следует анализировать DOM. Сделать это можно с помощью команды того же меню Inspect Element (просмотреть код).

Различия между исходным HTML-кодом, полученным с сервера, и DOM

Исходный HTML-код, полученный с сервера (то, что выводится по команде View page source) — это нечто вроде кулинарного рецепта. Он предоставляет информацию о том, какие ингредиенты входят в состав некоего блюда, содержит инструкции по приготовлению. Но рецепт — это не готовое блюдо.

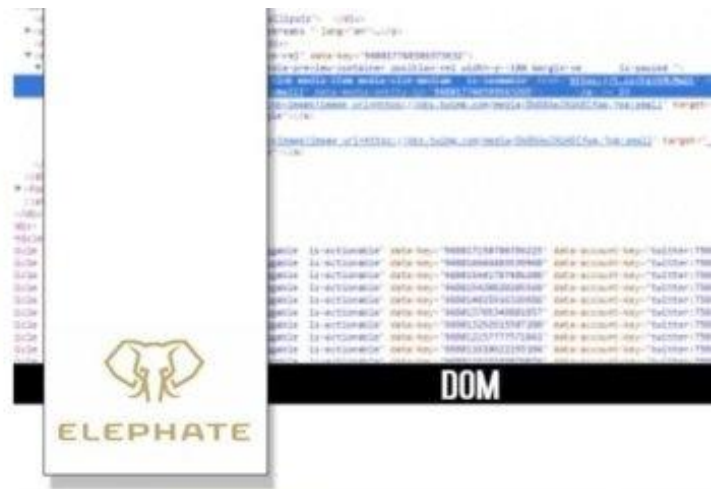
DOM (команда Inspect Element) — это блюдо, приготовленное по «HTML-рецепту». В самом начале браузер загружает «рецепт», потом занимается приготовлением блюда, и уже в итоге, после того, как страница полностью загружена, перед нами появляется

нечто «съедобное».



(right click -> View page source)

It is just a cooking recipe.
It provides information about what ingredients you should use to bake a cake.
It contains a set of instructions.
But it's not the actual cake.



(right click -> inspect element)

It is the actual baking of the cake.
In the beginning, it's just a recipe (an HTML document) and then, after sometime it gains a form and then it's baked (page fully loaded).

HTML, загруженный с сервера, и DOM

Обратите внимание на то, что если Google не удаётся сформировать страницу на основе загруженного HTML-кода, он может просто проиндексировать этот исходный HTML-код (который не содержит динамически обновляемого содержимого).

Подробности об этом можно найти в [данном материале](#) Барри Адамса. Барри, кроме того, даёт советы, касающиеся того, как быстро сравнить исходный HTML и DOM.

Итоги

В этом материале мы поговорили о том, как Google обрабатывает

сайты, которые созданы с применением JavaScript-технологий. В частности, огромное значение имеет то, что для анализа страниц в Google используется Chrome 41. Это накладывает определённые ограничения на применение JavaScript.

Во второй части перевода этого материала речь пойдёт о том, на что стоит обратить внимание для того, чтобы JS-сайты нормально индексировались поисковыми системами. Там же будут подняты ещё некоторые темы, касающиеся SEO и JavaScript.

Уважаемые читатели! Как вы анализируете свои сайты, проверяя, индексирует ли их Google так, как вы этого ожидаете?

Habrahabr10

Промо-код для скидки в 10% на наши виртуальные сервера

Проголосовать:

↑

+30

↓

Поделиться:

f

🐦

vk

Сохранить:

📌

Комментарии (5)

Похожие публикации

Неявное преобразование типов в JavaScript. Сколько будет !+[]+[]+![]?

ПЕРЕВОД

ru_vds • 30 января в 14:13

29

JavaScript-прокси: и красиво, и полезно

ПЕРЕВОД

ru_vds • 29 января в 12:23

13

JavaScript ES8 и переход на async / await

ПЕРЕВОД

ru_vds • 10 октября 2017 в 14:58

107

Популярное за сутки

Наташа — библиотека для извлечения структурированной информации из текстов на русском языке

alexkuku • вчера в 16:12

14

Unit-тестирование скриншотами: преодолеваем звуковой барьер. Расшифровка доклада

lahmatiy • вчера в 13:05

4

Люди не хотят чего-то действительно нового — они хотят привычное, но сделанное иначе

ПЕРЕВОД

25

Руководство по SEO JavaScript-сайтов. Часть 2. Проблемы, эксперименты и рекомендации

ПЕРЕВОД

ru_vds • вчера в 12:04

2

Как адаптировать игру на Unity под iPhone X к апрелю

P1CACHU • вчера в 16:13

0

Лучшее на Geektimes

Стивен Хокинг, автор «Краткой истории времени», умер на 77 году жизни

HostingManager • вчера в 13:49

33

Обзор рынка моноколес 2018

lozga • вчера в 06:58

70

«Битва за Telegram»: 35 пользователей подали в суд на ФСБ

alizar • вчера в 15:14

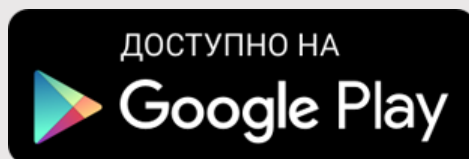
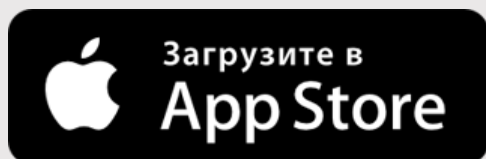
40

Стивен Хокинг и его работа — что дал ученый человечеству?

marks • вчера в 14:46

8

Мобильное приложение



Полная версия

2006 – 2018 © TM