12+

Иван Дискин

SEO
в 2020-е годы

Иван Дискин

SEO в 2020-е годы

«Издательские решения»

Дискин И. С.

SEO в 2020-е годы / И. С. Дискин — «Издательские решения»,

Поисковая система видит сайт иначе, чем человек. JavaScript, рендеринг, индексация, robots. txt, canonical, внутренняя перелинковка и структура URL напрямую влияют на то, что поисковый робот сможет обнаружить, получить и обработать. Эта книга разбирает техническую сторону SEO через архитектуру сайтов, диагностику проблем и практические кейсы. Отдельные главы посвящены изменениям поиска и появлению генеративных ответов.

Содержание

SEO в 2020-е годы

Иван Сергеевич Дискин

ISBN 978-5-0071-2792-9
Создано в интеллектуальной издательской системе Ridero

SEO в 2020-е годы
Иван Дискин
Москва
2026

Оглавление

Часть I. Архитектура современного поиска

Глава 1. Поисковая система как вычислительная среда

Страница, которую видит пользователь, и документ, который получает поисковая система, — не одно и то же.

Пользователь открывает URL и видит готовую страницу: заголовок, текст, изображения, цены, кнопки, отзывы. Всё это появляется мгновенно, как будто страница просто существовала где-то на сервере и её оставалось только показать.

Поисковая система проходит через совсем другой процесс. Сначала нужно обнаружить URL — узнать, что он вообще существует. Затем получить его содержимое — не обязательно с первого раза и не обязательно в том же виде, в каком его получит браузер пользователя. Затем, если это необходимо, выполнить JavaScript — это отдельный этап обработки, который требует дополнительных ресурсов и не гарантирует получения того же результата, который видит пользователь. Извлечь текст и ссылки из результата. Определить, какую версию страницы считать канонической, если таких версий несколько. Решить, стоит ли вообще включать документ в индекс — и это решение может быть отрицательным даже для полностью корректной страницы. И только после всего этого, сопоставив документ с другими документами и с запросами пользователей, у поисковой системы появляется основание для того, чтобы вообще думать о позиции в выдаче.

Именно здесь начинается современное SEO. Не с позиции. С вопроса, доживёт ли документ до момента, когда позиция вообще станет иметь смысл.

Сайт, который выглядел исправным

В одном из технических аудитов, которые мне довелось разбирать, сайт с точки зрения пользователя выглядел полностью здоровым. Страницы открывались без задержек, контент был на месте, навигация работала, форма обратной связи отправлялась, каталог листался. Ни один человек, тестирующий сайт руками, не нашёл бы повода для жалобы.

Поисковая система видела этот сайт иначе.

Часть страниц она обнаруживала — но не индексировала, без явной ошибки, а просто не включая их в индекс, причём причина такого решения не всегда очевидна владельцу сайта. На некоторых URL содержимое, полученное после выполнения JavaScript, отличалось от того, что видел пользователь в браузере: часть блоков подгружалась по клику или при прокрутке, и поисковый робот не получал их в ходе обработки страницы. Xml-карта сайта (sitemap. xml) сообщала поисковику один набор страниц — актуальный, свежий, аккуратно сгенерированный. Внутренняя перелинковка сайта вела к другому набору. А часть технических сигналов на одних и тех же URL прямо противоречила друг другу: canonical указывал на одну версию, а hreflang — на другую структуру URL.

Ни один из этих фактов сам по себе не был катастрофой. И нельзя сказать, что сайт «плохо оптимизировали» — в привычном смысле, будто где-то забыли прописать тег или неправильно расставили ключевые слова. Проблема была глубже: сайт и поисковая система по-разному понимали, что именно является страницей. У сайта было одно представление о соб-

ственной структуре. У поисковой системы — другое. И работа с каждым из них по отдельности ничего бы не исправила, потому что несовпадение возникало именно на стыке.

SEO начинается раньше, чем принято думать

Отсюда можно сформулировать тезис, на котором держится вся эта книга: SEO начинается не с позиции в выдаче. SEO начинается с того, может ли поисковая система правильно увидеть, понять и выбрать документ, который компания хочет показать пользователю. Все последующие этапы — оценка содержания, ссылочных сигналов, релевантности и ранжирование — зависят от того, насколько корректно поисковая система получила и обработала документ.

Долгое время это условие выполнялось как бы само собой. Сайт в основном представлял собой набор HTML-документов, каждый со своим URL. Сервер отдавал готовый документ, а поисковому роботу в большинстве случаев было достаточно получить его и разобрать разметку. Расхождение между «что видит пользователь» и «что получает поисковик» было минимальным, почти нулевым, и именно поэтому о нём никто особенно не думал.

Это условие перестало выполняться незаметно — не в какой-то один момент, а постепенно, по мере того как веб-страницы всё чаще стали работать как приложения. Сейчас достаточно сказать, что разрыв возник — механику и историю этого перехода я разберу в следующей главе. Здесь важнее зафиксировать другое: этот разрыв не устранён и вряд ли будет устранён окончательно. Поисковые системы стали лучше справляться с современным вебом, чем десять лет назад — это можно наблюдать на практике, хотя точную величину этого прогресса я бы не стал оцифровывать: слишком по-разному ведут себя разные краулеры на разных типах сайтов, чтобы говорить об этом одной цифрой. Но справляться лучше не значит справляться полностью и не значит справляться предсказуемо.

Поисковая система как вычислительная среда

Здесь полезно сменить угол зрения на саму поисковую систему. Привычная модель — поисковик как большой каталог, картотека страниц, которую нужно правильно заполнить, — работала, пока страница была статичным объектом. Для современного веба эта модель обманчива. Точнее говорить о поисковой системе как о вычислительной среде: она не просто хранит документы, а постоянно обрабатывает их: обнаруживает, получает, при необходимости рендерит, интерпретирует, сопоставляет с другими сигналами и принимает решения об индексации. Это не единоразовое действие «добавить в базу», а процесс, который может повторяться, откладываться и зависеть от множества факторов, в том числе от доступных поисковой системе ресурсов.

Это смещение — от документа к вычислению — не терминологическая тонкость. Если страница просто хранится, то ошибка — это ошибка в самой странице: опечатка в теге, неправильный статус-код, забытый noindex. Такие ошибки локальны, их видно и легко исправить. Если же представление страницы формируется в процессе обработки, то ошибка может возникать на стыке этапов — между тем, что отдаёт сервер, и тем, что получается после рендеринга; между тем, что заявлено в sitemap. xml, и тем, что реально содержит внутренняя структура; между разными версиями одного URL. Такие проблемы не всегда локальны. Их не видно при обычном просмотре сайта — ни пользователем, ни даже специалистом, который открывает страницы в браузере и не находит поводов для жалоб, как это было в примере выше. Их можно увидеть только тогда, когда смотришь на сайт не как на набор страниц, а как на процесс их обнаружения, обработки и выбора поисковой системой.

Не могу сказать, что эта модель была для меня очевидной сразу. Первые несколько лет работы с техническим SEO я, как и многие другие специалисты, диагностировал сайты по

чек-листу: заголовки, мета-теги, дубли, скорость загрузки, — и такой чек-лист действительно позволял находить большую часть типовых проблем на сайтах того времени. На современных сайтах такой чек-лист тоже позволяет находить часть проблем, но перестаёт объяснять некоторые расхождения вроде описанного выше — там, где по основным пунктам чек-листа всё выглядит корректно, а страница всё равно не индексируется. Именно такие случаи постепенно заставили меня пересмотреть модель — не потому, что старая была неправильной, а потому, что со временем её стало недостаточно.

Что это меняет

Если принять эту рамку, то работа с технической стороной сайта перестаёт быть про «правильную разметку одной страницы» и становится про согласованность между несколькими независимыми источниками информации о сайте: тем, что отдаёт сервер, тем, что видно после выполнения JavaScript, тем, что заявляет sitemap, тем, что подтверждает внутренняя перелинковка, тем, что говорят canonical и hreflang. Ни один из этих источников нельзя считать главным. Проблема часто возникает не внутри одного из них, а в зазоре между ними.

Дальше в этой книге я буду возвращаться к этой рамке — при разговоре про рендеринг, про жизненный цикл URL-адреса, про краулинговый бюджет, про структурированные данные. Но начать нужно было именно с неё, а не с истории SEO или перечисления фреймворков, потому что без этого сдвига в понимании остальные главы легко читаются как список инструментов — а это не список инструментов, которые нужно последовательно освоить. Это попытка объяснить, почему техническое SEO вообще стало настолько сложной дисциплиной: мы больше не работаем только с документом, который поисковая система просто скачивает. Работаем мы с системой, которая должна этот документ обнаружить, обработать, интерпретировать и решить, стоит ли включать его в индекс. А значит, следующий вопрос — как мы вообще пришли к ситуации, в которой страница перестала быть просто документом.

Глава 2. Когда страницы стали приложениями

Разработчик, который в 2008 году делал сайт, и разработчик, который в 2015 году делал «сайт», решали разные задачи — хотя формально работали с одним и тем же словом.

Первый собирал документы. Пользователь запрашивал URL, сервер формировал HTML — иногда на лету, из шаблона и данных из базы, но результатом серверного запроса всё равно был готовый HTML-документ — и сервер отправлял его браузеру. Переход на следующую страницу обычно означал новый запрос и повторную загрузку документа. Это было медленно, дёргано, с белым экраном между кликами — но при этом содержимое было непосредственно представлено в HTML и доступно браузеру, поисковому роботу и человеку, изучающему исходный код.

Второй собирал приложение. Первый запрос по-прежнему уходил на сервер, но в некоторых архитектурах в ответ приходил не готовый документ, а минимальный HTML-каркас и ссылка на JavaScript-код, который должен был сформировать остальную часть интерфейса уже в браузере: отрисовать интерфейс, запросить данные отдельным вызовом, обновить содержимое без перезагрузки страницы. Переход между разделами сайта переставал быть переходом между документами — он становился изменением состояния внутри одного приложения, которое внешне могло выглядеть как переход между страницами.

Почему это вообще понадобилось

У этого сдвига были вполне конкретные причины, и поисковые системы к ним отношения не имели. Веб-интерфейсы усложнялись — почта, карты, соцсети, панели администрирования — и постоянная перезагрузка всей страницы ради обновления одного блока стала выглядеть архитектурным анахронизмом. Технология, которая сделала такой подход возможным, появилась раньше, чем может показаться: XMLHttpRequest существовал уже в конце девяностых, но особенно заметным подход стал после появления Gmail в 2004 году, который показал возможности обновления интерфейса без полной перезагрузки страницы.

Дальше идея постепенно распространилась по вебу — сначала точечно, отдельными интерактивными блоками на в остальном статичных страницах, затем всё смелее, пока не оформилась в отдельный класс архитектуры — single-page application, SPA: приложение, в котором основная часть интерфейса формируется на стороне клиента, а сервер может отвечать преимущественно за данные, а не за готовую разметку. AngularJS, впервые выпущенный Google в 2010 году, стал одним из первых массовых фреймворков, закрепивших такой подход. Это был самостоятельный проект, а не ранняя версия современного Angular, но именно вокруг AngularJS во многом сформировалась привычка строить веб-интерфейсы как приложения. React, впервые опубликованный в 2013 году, закрепил эту модель ещё сильнее и сделал компонентный подход к интерфейсам массовым.

С точки зрения продукта и разработки это было настоящим прогрессом, и я не хочу делать вид, что предпочёл бы вернуться назад. Интерфейсы стали отзывчивее. Границы между «сайтом» и «приложением» во многих случаях стали менее определёнными — что для многих продуктов было именно тем, что требовалось. Фронтенд-разработка отделилась в собственную дисциплину со своими инструментами, тестами, экосистемой пакетов. Всё это правда.

Побочный эффект, который редко попадал в повестку

Правда и то, что вопрос о том, как новая архитектура будет обнаруживаться и обрабатываться поисковыми системами, далеко не всегда учитывался при её выборе — не потому

что о нём буквально никто не думал: отдельные SEO-специалисты и команды поднимали его вполне активно, — а потому что он редко был среди определяющих факторов. Решение выбиралось инженерной командой исходя из скорости разработки, удобства интерфейса и доступности специалистов — по вполне рациональным для продукта причинам. SEO при этом могло вообще не рассматриваться как отдельное требование.

А для поисковой системы контентная модель поменялась радикально. Раньше документ приходил цельным — HTML со всем содержимым внутри, который можно было скачать и разобрать сразу. Теперь в некоторых архитектурах по тому же URL приходила минимальная оболочка, а значительная часть контента появлялась только после выполнения JavaScript — то есть требовал не только загрузки HTML, но и дополнительной обработки: выполнения JavaScript, загрузки необходимых ресурсов, запросов к API и формирования итогового состояния страницы. Обработка одного URL, которая раньше во многих случаях сводилась к получению готового документа, стала включать существенно больше шагов: получить оболочку, загрузить необходимые ресурсы, выполнить JavaScript, дождаться необходимых сетевых запросов и сформировать итоговое состояние страницы. Не всегда именно в такой последовательности и не всегда с одинаковой ценой — современные системы умеют часть этого процесса оптимизировать и кэшировать, — но сама обработка стала многоступенчатой там, где раньше во многих случаях была существенно проще.

Одна из самых показательных иллюстраций этого разрыва — документированная история конца 2000-х и начала 2010-х. В 2009 году Google предложил индустрии временное соглашение для индексации AJAX-контента: сайты могли использовать в URL «hash-bang» — site.com/#!/page — и параллельно отдавать краулеру по специальному адресу (?_escaped_fragment_=…) заранее отрисованную HTML-версию той же страницы. Схема работала в браузере как обычная client-side маршрутизация, а для поисковика требовала отдельного, недёшево поддерживаемого механизма подготовки статичных копий. Twitter одно время использовал hash-bang URL в собственном интерфейсе — и в мае 2012 года публично объявил об отказе от них и переходе к History API и pushState (), связывая это в том числе с улучшением скорости загрузки. Сам же Google отказался от предложенной им схемы позже и более концептуально: в октябре 2015 года компания официально депрекировала AJAX crawling scheme, заявив, что Googlebot к этому моменту уже мог обрабатывать JavaScript-страницы значительно эффективнее, чем раньше — поэтому необходимость в отдельном протоколе постепенно исчезла.

Здесь я намеренно не привожу цифру вроде «столько-то процентов SPA-сайтов теряли трафик» — такой статистики, которой я бы доверял, я не нашёл, а множество случайных цифр из блогов той эпохи не выдерживают проверки на источник. Достаточно того, что сама категория проблемы была настолько распространена, что вокруг неё выросла отдельная профессиональная тема — «SEO для JavaScript-сайтов» — и отдельный набор паттернов, часть из которых, включая hash-bang, впоследствии перестала использоваться как основной подход.

Индустрия не стояла на месте

Реакция на этот разрыв не была единой и не появилась мгновенно. Одни команды пытались добиться того, чтобы поисковая система получала примерно то же содержимое, которое видел браузер, — через предварительный рендеринг и специальные сервисы, отдававшие поисковому роботу заранее сформированную версию страницы. Другие начали пересматривать саму архитектуру рендеринга: если проблема в том, что HTML собирается в браузере пользователя, так почему бы не собирать его раньше — на сервере, до того как запрос вообще дойдёт до клиента?

Из этой развилки выросло то, что сегодня называют разными моделями рендеринга —серверный рендеринг (server-side rendering, SSR), генерация статических страниц (static site generation, SSG), инкрементальная генерация статических страниц (incremental static regeneration, ISR), гибридные схемы, где разные части сайта могут использовать разные модели рендеринга. Это не история фреймворков ради истории фреймворков: это история того, как индустрия несколько раз подряд пересматривала один и тот же вопрос — где именно и в какой момент документ, который в итоге увидит пользователь и попытается разобрать поисковик, должен формироваться. Этому вопросу и тому, что каждый из ответов на него значит для SEO, посвящена отдельная глава — четвёртая.

Но прежде чем переходить к моделям рендеринга, стоит остановиться на том, с чем именно сталкивается поисковый робот, когда встречает JavaScript-сайт сегодня — не в 2012 году, а сейчас. Потому что за прошедшее десятилетие поисковые системы тоже не стояли на месте, и разрыв, о котором шла речь в этой главе, сузился — но не закрылся. Где именно проходит его нынешняя граница — тема следующей главы.

Глава 3. JavaScript и пределы поискового краулинга

«Google давно умеет рендерить JavaScript, так что это больше не проблема» — фраза, которую я слышал десятки раз, обычно от разработчиков, которые искренне не понимают, почему SEO-специалист продолжает задавать вопросы про рендеринг. И в этой фразе есть правда. Проблема в том, что правда в ней — примерно наполовину.

Google действительно умеет обрабатывать JavaScript и рендерить страницы с его использованием. С 2019 Googlebot использует регулярно обновляемую (evergreen) версию Chromium, которая регулярно обновляется вместе с актуальным Chrome — это завершило длительный период, когда Googlebot работал с устаревшим движком рендеринга и не поддерживал часть современного синтаксиса и API. Сегодняшний Googlebot в этом смысле значительно ближе к современному браузеру, чем несколько лет назад. Но «умеет рендерить» и «рендерит сразу, для каждой страницы, без потерь» — не одно и то же. Именно в зазоре между этими двумя утверждениями и живёт большая часть современных проблем JavaScript-SEO.

Между обходом и индексацией есть ещё один этап

По официальному описанию Google, обработка страницы состоит из нескольких последовательных этапов: обход, рендеринг и индексация — не одно действие, а конвейер. Сначала Googlebot получает HTML-ответ сервера и обрабатывает содержащиеся в нём ссылки, текст, метаданные и другие доступные без выполнения JavaScript данные. Если контент уже присутствует на этом этапе — например, страница отрисована на сервере, — индексация может произойти прямо здесь, без дополнительных шагов. Если же значимая часть контента появляется только после выполнения JavaScript, страница попадает в очередь на рендеринг, и лишь когда до неё доходит очередь, специальный сервис Google, Web Rendering Service, обрабатывает её в среде на базе Chromium, выполняет код, выполняет необходимые для обработки сетевые запросы и передаёт получившийся DOM обратно на индексацию.

В индустрии эту последовательность часто называют «двумя волнами индексации» — но стоит сразу оговориться: это не термин самого Google, а удобное разговорное обозначение, которое прижилось среди практиков. Сути это не меняет: между первичной обработкой HTML и последующим рендерингом существует временной разрыв, и этот разрыв не гарантированно мал. В официальной документации Google формулировка сознательно расплывчата: страница может находиться в очереди на рендеринг некоторое время, а иногда и дольше. Я бы не стал переводить это в конкретные часы или дни — слишком по-разному это выглядит на практике в зависимости от масштаба сайта и текущей загрузки инфраструктуры Google, а более точных универсальных сроков сам Google не публикует. Но сам факт очереди — не гипотеза, а задокументированное поведение системы.

Где именно сегодня возникают ограничения

Если раньше главная проблема звучала грубо — «Google вообще не видит JS-контент», — то сегодняшняя картина стала значительно сложнее и не менее важна: рендеринг в целом работает, но обработка конкретной страницы может завершиться без получения части нужного содержимого по вполне предсказуемым причинам.

Первая — зависимость от времени и ресурсов рендеринга. Если контент появляется только после длинной цепочки последовательных сетевых запросов — сначала загрузить конфигурацию, потом по ней запросить список, потом по каждому элементу списка отдельный вызов, — результат рендеринга может оказаться неполным: обработка страницы может завер-

шиться раньше, чем цепочка запросов будет полностью выполнена. Google прямо указывает, что рендеринг может быть отложен и зависит от доступных ресурсов, поэтому рассчитывать на фиксированный срок обработки страницы нельзя.

Вторая — отсутствие пользовательского поведения. Google не рассчитывает на обычное пользовательское взаимодействие со страницей: не следует считать клики, прокрутку или наведение курсора гарантированным способом загрузки критичного содержимого. Контент, который появляется по этим действиям — «ленивая» подгрузка по скроллу, раскрывающиеся по клику блоки, — поисковая система может не получить, если он не реализован так, чтобы срабатывать и без событий пользователя.

Третья — статус-коды и директивы, которые могут повлиять на дальнейшую обработку страницы. Здесь у меня будет отсылка чуть вперёд по времени, чем обычно уместно в этой главе, но она важна: в декабре 2025 года Google уточнил в документации, что страницы с некоторыми не-200 статус-кодами, например страницы с ответом 404, могут не проходить этап рендеринга. Для JavaScript-сайтов это особенно важно при работе с ошибками и «мягкими» 404: если сервер возвращает ошибочный статус-код, не стоит рассчитывать, что последующий JavaScript-рендеринг изменит решение поисковой системы об обработке страницы.

С noindex ситуация похожая, но здесь особенно важно не делать слишком широких выводов: если директива noindex присутствует уже в исходном HTML, Google может не выполнять дальнейший рендеринг страницы. Это особенно важно для JS-сайтов: рассчитывать на то, что JavaScript позже удалит или изменит исходный noindex, не стоит — решение может быть принято раньше, чем этот код вообще выполнится.

етвёртая, и одна из самых распространённых, — заблокированные ресурсы. Файл robots. txt, запрещающий поисковому роботу доступ к файлам JavaScript или CSS, необходимых для построения страницы, может лишить рендерер ресурсов, необходимых для корректного построения и понимания страницы: он получает доступ к HTML-странице, но не может загрузить часть ресурсов, необходимых для её формирования. Об этом предупреждают едва ли не в каждом гайде по JavaScript SEO уже лет десять — и тем не менее эта ошибка продолжает встречаться на реальных аудитах с завидной регулярностью, что скорее говорит не о недостатке информации, а о том, что проверка robots. txt на блокировку критичных ресурсов до сих пор не всегда входит в технический аудит наравне с проверкой доступности самих страниц.

Наконец, доступность ресурса ещё не означает, что Google обязательно будет его загружать. Google прямо указывает, что Googlebot и Web Rendering Service могут не запрашивать ресурсы, которые не считает необходимыми для обработки основного содержимого страницы. Для разработчика это важное различие: «ресурс доступен поисковику» и «поисковик обязательно загрузит этот ресурс при обработке страницы» — не одно и то же. Нет смысла полагаться на первое как на гарантию второго.

Не только Google

Отдельная тема, которую легко упустить из виду, если думать только о позициях в Google, — остальные потребители контента сайта. Bing также развивал возможности обработки JavaScript, но эти возможности не полностью совпадают с возможностями Google. Нет оснований считать, что если сайт хорошо индексируется в одной системе, то так же будет в другой без отдельной проверки. А у других поисковых и AI-систем архитектура обхода может отличаться от Google ещё сильнее. Поэтому корректно индексируемый Google JavaScript-сайт не стоит автоматически считать одинаково доступным для всех остальных краулеров — особенно осторожно стоит относиться к системам, для которых приоритетны быстрый массовый сбор и предварительная обработка контента. Подробно развивать здесь эту тему я не буду — она уже выходит за рамки классического поиска и связана в том числе с генеративным поис-

ком, и заслуживает отдельного, доказательного разговора в другой книге. Но для полноты картины важно проговорить: техническая база, о которой эта книга, не заканчивается на одном Googlebot, даже если именно на нём чаще всего сосредоточено внимание.

Что из этого следует

Разрыв, о котором шла речь в прошлой главе, действительно сузился — Google в 2026 году заметно лучше справляется с современным вебом, чем в начале 2010-х, и это видно не только из официальной документации, но и из практики. Но он не закрылся и, судя по тому, как устроена сама архитектура — с очередью, зависимостью от доступных ресурсов, статус-кодов и управляющих директив, — вряд ли закроется полностью: слишком много условий должно выполняться одновременно, чтобы можно было гарантировать одинаково полный результат рендеринга для каждой страницы.

Это совсем не значит, что JavaScript нужно избегать. Современный веб уже устроен иначе, и клиентский код стал нормальной частью большинства сложных сайтов. Вопрос в другом: насколько критичен для сайта результат выполнения этого кода поисковой системой.

Если поисковая система должна выполнить несколько скриптов, загрузить дополнительные ресурсы, дождаться запросов к API и только после этого получить основной контент страницы, SEO начинает зависеть от всей этой цепочки. Чем она сложнее, тем больше в ней точек, в которых результат может отличаться от того, что увидел пользователь.

Поэтому выбор архитектуры рендеринга имеет прямое отношение к SEO. Не потому, что одна модель сама по себе «хороша для SEO», а другая «плоха», а потому, что разные модели по-разному распределяют работу между сервером, браузером и поисковой системой. И это уже нужно учитывать при проектировании сайта, а не пытаться исправить после его запуска. Какие архитектурные модели рендеринга возникли в ответ на эту проблему и что каждая из них меняет в отношениях сайта с поисковой системой, разберём в следующей главе.

Глава 4. Рендеринг как часть SEO-архитектуры

Один и тот же интернет-магазин можно устроить по-разному. В одном случае сервер сразу отдаёт готовый HTML. В другом браузер получает минимальную оболочку и формирует страницу сам. Можно заранее подготовить HTML для всех нужных страниц во время сборки сайта, а можно сочетать эти подходы и обновлять отдельные страницы по мере изменения данных.

Для пользователя разница между этими вариантами часто почти незаметна. Он открывает карточку товара, видит название, цену и фотографии, нажимает «В корзину» и продолжает работу с сайтом. Для поисковой системы разница принципиальная: в зависимости от архитектуры она может получить готовый документ сразу или сначала получить только его основу, а затем потратить дополнительные ресурсы на то, чтобы сформировать итоговое содержимое.

Поэтому вопрос рендеринга для SEO заключается не в выборе «правильного» фреймворка и даже не в том, нужно ли использовать JavaScript. Гораздо важнее другое: на каком этапе обработки страницы поисковая система получает тот HTML и то содержимое, которые мы хотим видеть в поиске.

От ответа на этот вопрос зависит, насколько сайт будет зависеть от выполнения JavaScript, доступности дополнительных ресурсов, времени обработки и других условий, которые находятся уже не на стороне самого HTML-документа. Поэтому рендеринг стоит рассматривать как часть SEO-архитектуры сайта. Причём решение может быть разным для разных типов страниц и даже для разных частей одной страницы.

Начнём с самого зависимого от браузера варианта.

SSR: документ появляется на сервере

Рендеринг на стороне сервера (server-side rendering, SSR) переносит формирование HTML на сервер: при обращении к странице сервер получает необходимые данные, собирает HTML и отдаёт его уже с основным содержимым. Для поисковой системы это принципиальная разница: уже на первом этапе обработки она получает содержательный HTML, а не минимальную оболочку, которую ещё предстоит собрать с помощью JavaScript.

У этой модели есть и обратная сторона, о которой при разговоре об SEO говорят заметно реже. HTML приходится формировать при обращении к странице. Серверу нужно получить данные, собрать разметку и отправить результат пользователю или поисковому роботу. Если таких запросов много, нагрузка быстро становится существенной: поисковые роботы могут последовательно запрашивать тысячи и десятки тысяч страниц. Поэтому на практике SSR часто требует продуманного кэширования на сервере или CDN, иначе одни и те же страницы приходится заново формировать при каждом обращении.

Есть и вопрос скорости ответа. Само по себе наличие готового HTML ещё не означает, что страница будет быстро получена: если сервер долго формирует документ или ждёт данные из нескольких источников, преимущество SSR частично теряется. Для SEO это важно не потому, что поисковая система ставит отдельный «балл за SSR», а потому, что скорость ответа влияет и на пользовательский опыт, и на эффективность обхода сайта.

При этом SSR не превращает сайт автоматически в хорошо оптимизированный. Он решает одну конкретную задачу: позволяет отдать основное содержимое уже в первоначальном HTML. Но остаются дублирующиеся URL, ошибки в canonical, проблемы с перелинковкой, слабый или нерелевантный контент и множество других факторов.

Поэтому было бы ошибкой противопоставлять хорошо реализованный SSR любому варианту клиентского рендеринга. Сайт на SSR с плохо устроенной архитектурой может работать хуже для поиска, чем аккуратно спроектированный сайт на CSR с предварительным рендерингом. Модель рендеринга задаёт условия, в которых работает SEO, но сама по себе его не определяет.

SSG: документ появляется до запроса

Статическая генерация страниц (static site generation, SSG) формирует HTML заранее, во время сборки сайта, а не в момент обращения к странице. В результате получается набор готовых файлов, которые можно отдавать пользователям напрямую, в том числе через CDN, без обращения к серверу приложения при каждом запросе. Это позволяет добиться высокой скорости ответа и существенно снизить нагрузку на серверную инфраструктуру.

SSG особенно хорошо подходит для сайтов, где содержимое страниц меняется относительно редко: блогов, документации, справочных разделов, каталогов со стабильными данными. Все необходимые данные известны на момент сборки, поэтому основной контент, метаданные, canonical и структурированные данные уже присутствуют в готовом HTML. Поисковой системе не нужно ждать отдельного этапа рендеринга, чтобы получить это содержимое.

Главное ограничение SSG становится заметно там, где данные меняются часто. Например, каталог из ста тысяч товаров с ценами и остатками, которые обновляются несколько раз в день, трудно поддерживать в актуальном состоянии при чистом SSG. Если запускать полную пересборку после каждого изменения, это быстро становится слишком затратным. Если же собирать сайт реже, часть страниц какое-то время будет содержать устаревшие данные.

Та же проблема возникает с содержимым, которое зависит от конкретного пользователя. Персональные рекомендации, данные авторизованного пользователя или другие индивидуальные элементы нельзя заранее собрать в одной общей версии страницы. SSG хорошо работает там, где страницу можно подготовить заранее и отдавать одинаковой или почти одинаковой для всех посетителей.

ISR: документ появляется заранее, но может обновляться

Именно из этого ограничения выросла модель, которую Next. js представил в 2020 году под названием инкрементальная регенерация статических страниц (incremental static regeneration, ISR). Её идея проста: страница по-прежнему формируется как статическая, но уже не обязательно один раз при полной сборке сайта. Отдельную страницу можно заново сформировать после истечения заданного времени или по внешнему событию, не пересобирая при этом весь сайт. Для большого каталога это существенно меняет экономику обновлений: изменение данных одной страницы не требует заново формировать тысячи остальных.

При этом пользователь обычно получает уже подготовленную версию страницы. Обновление может происходить отдельно от его запроса, поэтому актуализация данных не обязательно должна увеличивать время ответа.

Для SEO эта модель важна прежде всего тем, что позволяет сочетать два свойства. Как и при SSG, поисковая система получает готовый HTML без необходимости сначала выполнять JavaScript. Но содержимое страницы при этом можно регулярно обновлять без полной пересборки сайта.

Именно такие модели постепенно разрушили простое противопоставление «SSR против CSR», которое долго использовалось для описания архитектуры веб-приложений. На практике один сайт может использовать разные способы формирования страниц в зависимости от того,

насколько часто меняются данные, насколько важна скорость ответа и требуется ли персонализация.

Поэтому вопрос уже не в том, какую модель выбрать для сайта целиком. Гораздо важнее определить, какая модель подходит каждому типу страниц и какие части страницы действительно должны формироваться на сервере, заранее или в браузере.

Гибридная архитектура: рендеринг как свойство страницы, а не сайта

Один и тот же сайт сегодня может использовать несколько моделей рендеринга одновременно. Например, статьи блога удобно заранее формировать при сборке сайта с помощью SSG, карточки товаров обновлять через ISR, а персонализированные разделы отдавать с сервера. Для отдельных интерактивных элементов, содержание которых не имеет значения для поиска, вполне достаточно CSR.

Причём разные модели могут использоваться не только для разных типов страниц, но и внутри одной страницы. Основной текст карточки товара может быть сформирован на сервере и попасть в HTML сразу, а блок рекомендаций загрузиться в браузере после открытия страницы. Для пользователя это одна страница, но с точки зрения её формирования в ней участвуют разные механизмы.

Поэтому сегодня уже не совсем правильно говорить, что сайт работает на CSR, SSR или SSG. Правильнее смотреть на конкретную страницу и на отдельные части её содержимого. Для SEO это особенно важно: далеко не весь JavaScript на странице представляет одинаковую ценность. Одно дело, когда в браузере появляется дополнительная кнопка или интерактивный фильтр, и совсем другое, когда только после выполнения JavaScript становится доступен основной текст, цена товара или список категорий.

Именно поэтому выбор модели рендеринга стоит рассматривать как часть архитектуры конкретного контента. Чем важнее элемент для поискового трафика, тем меньше оснований оставлять его появление исключительно на усмотрение браузера и последующего рендеринга поисковой системой.

Что SEO действительно должно знать о рендеринге

Из всего сказанного следует практический вывод. Он не сводится к совету «используйте SSR»: такой совет перестал быть универсальным, как только выбор модели рендеринга стал зависеть от конкретной страницы и её содержимого.

Вместо этого SEO-специалисту нужна другая привычка: для каждого важного элемента задавать конкретный вопрос — где и на каком этапе он появляется в документе, который в итоге получает поисковая система. Если элемент появляется только после выполнения JavaScript, это уже отдельное условие, которое нужно учитывать при оценке страницы.

Именно такой взгляд позволяет связать техническую архитектуру с реальным SEO: важно не название модели рендеринга само по себе, а то, какое содержимое поисковая система получает и на каком этапе обработки страницы.

Элемент	Что важно выяснить
Основной текст	На каком этапе появляется в HTML – сразу или только после выполнения JavaScript?
Тег <title>	Формируется на сервере или при сборке сайта либо дописывается скриптом на клиенте?
Мета-тег description	Присутствует в исходном HTML или появляется только после выполнения JavaScript?
Canonical	Доступен уже в исходном HTML или появляется только после рендеринга (то есть в отрисованном DOM)?
Hreflang	Задаётся статически или формируется динамически на стороне клиента?
Внутренние ссылки	Присутствуют ли как ссылки <a href> в HTML или DOM, или переходы реализованы только через обработчики JavaScript без отдельного URL?
Структурированные данные	На каком этапе генерируется JSON-LD – при сборке, на сервере или в браузере?
Товарные данные (цена, наличие)	Из какого источника и в какой момент они подгружаются – и насколько быстро устаревают?
Пагинация	Формируются ли отдельные индексируемые URL для страниц листинга, или переключение реализовано только через клиентское состояние?

Это не чек-лист для одноразовой проверки, а модель, к которой SEO-специалист возвращается при обсуждении новой функциональности или редизайна. Вопрос здесь в том, в какой момент и где появляется именно этот элемент. Разные ответы для разных элементов одной и той же страницы совершенно нормальны в 2026 году и сами по себе не означают, что архитектура устроена неправильно.

Итог

У рендеринга нет одной универсально правильной модели. Для одних страниц разумно заранее формировать готовый HTML, для других — собирать его на сервере при запросе, для третьих — сочетать разные подходы. Выбор зависит от того, какие данные содержит страница, как часто они меняются, нужна ли персонализация и насколько важно получить содержимое уже на первом этапе обработки.

Поэтому сам по себе выбор CSR, SSR, SSG или ISR ещё ничего не говорит о качестве SEO-архитектуры. Важно другое: понимает ли команда, где формируется содержимое страницы, когда оно становится доступным и что именно в этот момент получает поисковая система.

Проблема возникает тогда, когда эти вопросы вообще не входят в техническое проектирование. Например, команда может выбрать архитектуру, удобную для разработки и поддержки, а влияние этого решения на поисковую видимость обнаружить только после запуска. На этом этапе исправление может потребовать уже не настройки отдельных метатегов, а изменения способа формирования страниц и взаимодействия между сервером, браузером и данными.

Поэтому рендеринг стоит рассматривать как часть SEO-архитектуры с самого начала. SEO-специалисту важно участвовать в обсуждении таких решений не для того, чтобы навязы-

вать конкретную технологию, а для того, чтобы заранее определить, какое содержимое должно быть доступно поисковой системе и на каком этапе оно должно появляться.

На этом мы заканчиваем разговор о том, где и когда формируется документ. В следующей части разберём, что происходит с ним после этого: как поисковая система обнаруживает URL, получает страницу, обрабатывает её, принимает решение об индексации и в итоге определяет, какое место документ займёт в поиске.

Часть II. Что поисковик видит на самом деле

Глава 5. Жизненный цикл URL в поисковой системе

Владелец сайта обычно говорит: «страница проиндексирована» или «страница не индексируется», как будто индексация — это простое состояние, которое можно представить переключателем: включено или выключено. Для поисковой системы всё сложнее. URL не попадает в индекс одним действием. Сначала поисковой системе нужно узнать о его существовании, затем решить, когда его обходить, получить ответ от сервера, при необходимости обработать JavaScript, разобрать полученное содержимое и только после этого принять решение об индексации.

На любом из этих этапов обработка может быть отложена или завершиться без перехода к следующему. Причина не обязательно заключается в технической ошибке. Поисковая система может не считать дальнейшую обработку URL достаточно приоритетной, не получить нужные данные, обнаружить дублирующее содержимое или принять другое решение на основании доступных ей сигналов.

Эту последовательность полезно разложить на отдельные этапы. Не потому, что каждый из них требует самостоятельного технического разбора прямо сейчас: краулинговому бюджету, дублям, canonical и другим отдельным вопросам будут посвящены следующие главы. Такая карта нужна прежде всего для диагностики. Когда страница не появляется в поиске, важно понимать, на каком этапе возникло ограничение, иначе разные причины легко принять за одну и ту же проблему.

Здесь важно сделать ещё одну оговорку. В общих материалах Google описывает работу поисковой системы через несколько крупных процессов, среди которых обход, рендеринг и индексация. Более подробное деление, которое я буду использовать дальше, — обнаружение URL, планирование обхода, получение ответа, рендеринг, разбор содержимого и индексация — это упрощённая авторская модель для практической диагностики, а не официальное описание внутреннего процесса Google.

Реальный процесс не выглядит как строго последовательная конвейерная цепочка. Отдельные действия могут повторяться, откладываться или выполняться в рамках разных этапов обработки. Уже принятое решение также может быть пересмотрено при следующем обходе. Поэтому дальше эту схему стоит воспринимать именно как карту: она помогает определить, где искать причину проблемы, но не описывает внутреннюю реализацию поисковой системы до каждого технического шага.

Discovery: URL ещё не известен поисковой системе

Первый этап начинается раньше, чем можно подумать: прежде чем поисковая система сможет обработать страницу, ей нужно обнаружить её URL. Источников для этого несколько: ссылки с уже известных страниц, как внутренних, так и внешних, файлы sitemap, перенаправления со старых адресов и другие сигналы. В Search Console также можно отправить URL на проверку и запросить его индексацию, но это именно запрос на обработку, а не гарантия немедленного обхода или индексации.

Если URL ещё не обнаружен поисковой системой, она не может начать его обычную обработку. Не имеет значения, насколько качественным является размещённый на нём контент: сначала поисковой системе нужно узнать сам адрес.

Здесь появляется первая важная развилка, которую Google показывает в Search Console статусом «Discovered — currently not indexed». Это не сообщение о технической ошибке и не окончательный отказ от индексации. Оно означает, что Google уже знает о URL, но пока не выполнил его обход. Среди возможных причин Google указывает откладывание обхода, в том числе для того, чтобы не создавать чрезмерную нагрузку на сервер сайта.

Таким образом, даже обнаруженный URL не обязательно сразу переходит к следующему этапу. Между тем, что поисковая система уже знает об адресе, и фактическим обращением к нему может пройти некоторое время.

Crawl scheduling: когда выполнять обход

Когда URL уже известен, поисковая система принимает отдельное решение о том, когда имеет смысл обратиться к нему и сколько ресурсов выделить на обход.

В документации Google для описания этой задачи используются два понятия: crawl capacity и crawl demand. Crawl capacity показывает, насколько интенсивно Google может обращаться к сайту с учётом возможностей его серверной инфраструктуры. Если сервер отвечает медленно или начинает возвращать ошибки, допустимая частота запросов может снижаться. Crawl demand, в свою очередь, описывает потребность Google возвращаться к сайту и его URL. На неё влияют, в частности, важность и популярность содержимого, а также то, насколько оно изменилось с момента предыдущего обхода.

Из этого следует важный практический вывод: размер сайта сам по себе не определяет частоту его обхода. Небольшой сайт не обязан обходиться реже большого только потому, что на нём меньше страниц. Важны сочетание возможностей сервера и потребности поисковой системы получать или обновлять содержимое.

С новыми или малоизвестными сайтами я бы тоже не вводил отдельное правило вроде «Google специально занижает им приоритет». Такого универсального принципа Google не заявляет. У нового сайта просто может быть меньше накопленных сигналов, на основании которых поисковая система определяет потребность в его регулярном обходе. Поэтому обнаруженные URL такого сайта могут некоторое время оставаться необработанными.

Именно здесь находится одна из возможных причин длительного статуса «Discovered — currently not indexed». URL уже известен Google, но поисковый робот ещё не приступил к его обходу. Сам по себе этот статус не говорит о технической ошибке страницы.

Fetch: получение HTTP-ответа

Если URL выбран для обхода, Googlebot сначала должен определить, разрешено ли ему обращаться к нему с учётом правил robots. txt. Эти правила управляют доступом поисковых роботов к URL и ресурсам сайта. При этом robots. txt не является инструкцией «индексировать или не индексировать»: URL, закрытый от обхода, в некоторых случаях всё равно может появиться в результатах поиска без содержимого страницы.

Если обращение разрешено, Google получает HTTP-ответ. И здесь обработка ещё может остановиться. Сервер может вернуть ошибку, перенаправление или успешный ответ с кодом 200, в котором при этом практически нет нужного содержимого.

Статус ответа имеет значение и для дальнейшей обработки. В частности, Google отдельно уточняет, что страницы с кодом 200 передаются на рендеринг, тогда как для ответов с другими кодами HTTP такое поведение не гарантируется. Поэтому на этом этапе важно смотреть не только на сам URL, но и на то, что именно сервер вернул поисковому роботу.

Render: если необходимо

Если сервер уже отдал содержательный HTML, отдельный рендеринг для получения основного содержимого может быть не нужен. Для страниц, зависящих от JavaScript, ситуация другая: Google должен обработать скрипты и получить представление страницы, которое затем можно анализировать дальше.

Это именно тот этап, который мы подробно разбирали в предыдущих главах: выполнение JavaScript, необходимые сетевые запросы и формирование итогового DOM. При этом не стоит представлять рендеринг как обязательную вторую стадию для каждой страницы. Для обычного серверного HTML значительная часть информации уже доступна до него.

И ещё одна важная оговорка: результат рендеринга не следует считать раз и навсегда установленной версией документа. При следующем обходе Google может получить другое содержимое, увидеть изменения и заново оценить страницу. Поэтому здесь нас интересует не «финальный документ Google», а то представление страницы, которое поисковая система получила в рамках конкретной обработки.

Parse: что поисковая система извлекает из документа

После получения HTML, с выполнением рендеринга или без него, поисковая система должна разобрать содержимое документа. Она извлекает текст, ссылки, метаданные, структурированные данные и другие доступные элементы страницы.

Именно здесь становится особенно заметна разница между тем, что разработчик или редактор считает содержимым страницы, и тем, что поисковая система действительно получила в результате обработки. Текст может появляться только после выполнения JavaScript, ссылки могут отсутствовать в HTML, метаданные могут отличаться от ожидаемых, а часть данных может вообще не попасть в полученное представление страницы.

Это напрямую связано с проблемой, которую мы разбирали в первой главе: разные источники информации о сайте могут показывать разные картины. Сервер может отдавать одно содержимое, после рендеринга формируется другое, sitemap заявляет третье, а внутренняя структура сайта указывает на четвёртую версию.

На этом же уровне возникает вопрос о дублях и канонизации. Если поисковая система обнаруживает несколько URL с одинаковым или очень похожим основным содержимым, она может объединить их в одну группу и выбрать каноническую версию. Указанный сайтом canonical является одним из сигналов для этого решения, но не гарантирует, что Google выберет именно его.

Indexing: решение о включении в индекс

После получения и обработки документа поисковая система принимает решение о его индексации. Здесь оценивается не только техническая доступность страницы, но и само содержимое, его связь с другими версиями документа, качество и другие сигналы, которые используются в процессе индексации. Если страница дублирует уже известный документ, Google может выбрать другую URL как каноническую. Если содержимое недостаточно полезно или не соответствует требованиям поисковой системы, документ также может не попасть в индекс. Технические проблемы могут влиять на это решение, как и признаки спама или другие ограничения.

При этом важно не смешивать индексацию с отображением страницы в результатах поиска. Даже если документ включён в индекс, это ещё не означает, что он будет показан по

конкретному запросу, займёт высокую позицию или получит расширенное представление в выдаче.

Хороший практический пример снова даёт Search Console. Статус «Crawled — currently not indexed» означает, что Google уже выполнил обход страницы, но пока не включил её в индекс. Сам статус не означает окончательный отказ: страница может быть проиндексирована позже. Но он также не обещает, что это обязательно произойдёт. Поэтому читать его как «Google скоро проиндексирует страницу» неправильно. Это описание текущего состояния, а не прогноз.

Легко перепутать, но крайне нежелательно

Из всей этой цепочки стоит вынести три отдельных утверждения, которые на практике часто смешивают.

— «URL обойден» не означает «URL проиндексирован». Поисковая система могла получить страницу, но после обработки принять решение не включать её в индекс или выбрать друг9ой URL как канонический.

— «URL проиндексирован» не означает «URL хорошо ранжируется». Наличие документа в индексе означает, что поисковая система может рассматривать его для показа в результатах поиска. Позиция по конкретному запросу определяется уже другими сигналами и процессами.

— И третье, менее очевидное: «URL проиндексирован сейчас» не означает «URL останется проиндексированным». Индекс постоянно обновляется. При последующих обходах поисковая система может обнаружить изменения, новые дубли или другие сигналы и пересмотреть ранее принятое решение.

Зачем вообще нужна эта карта

Эта модель не заменяет документацию Google и тем более не претендует на описание его внутренней инфраструктуры. Её задача гораздо практичнее: дать точку опоры для диагностики.

Когда страница не появляется в поиске, вопрос «что не так с SEO» слишком широк. Вопрос «где сейчас находится эта страница в цепочке обработки и на каком этапе возникло ограничение» уже позволяет сформировать проверяемую гипотезу.

Если URL ещё не обнаружен, нужно искать проблему в его доступности для обнаружения. Если URL обнаружен, но долго не обходится, смотреть нужно уже на распределение ресурсов обхода. Если поисковый робот обращается к странице, но получает неожиданный ответ, проблема находится на уровне HTTP и серверной обработки. Если нужное содержимое появляется только после JavaScript, нужно проверять рендеринг. Если документ получен, но поисковая система не включает его в индекс, внимание смещается к содержимому, дублям, канонизации и другим сигналам индексации.

Именно с этой последней развилки мы переходим к следующему вопросу: сколько ресурсов поисковая система вообще готова выделить на обход большого количества URL и как этот ресурс распределяется между страницами одного сайта.

Как практическое обобщение, а не как техническое правило, можно сказать так: для сайта на пятьсот страниц объём обхода обычно не становится главным ограничением. Для сайта на пять миллионов URL распределение ресурса обхода уже может стать самостоятельной SEO-задачей. И вот здесь начинается разговор о краулинговом бюджете.

Глава 6. Краулинговый бюджет: что поисковик действительно готов обходить

Практически в каждом разговоре о краулинговом бюджете рано или поздно появляется фраза вроде «у Google есть N запросов в сутки на сайт». Обычно за ней следует конкретная цифра, которую кто-то когда-то где-то видел. Но фиксированного числа запросов для каждого сайта не существует.

Google распределяет ресурс обхода с учётом двух факторов: сколько запросов сайт способен нормально обрабатывать и насколько поисковой системе вообще нужно возвращаться к его страницам. Поэтому краулинговый бюджет правильнее воспринимать не как заранее выданный лимит, а как результат взаимодействия технических возможностей сайта и потребности поисковой системы в его обходе.

Как уже говорилось в предыдущей главе, Google описывает эти факторы через два понятия: crawl capacity (способность сайта выдерживать нагрузку от поискового робота) и crawl demand (потребность поисковой системы обходить страницы сайта). Именно их сочетание объясняет, почему одна и та же проблема почти незаметна для сайта на пятьсот страниц и становится серьёзной для сайта на пять миллионов URL.

Два фактора вместо фиксированного лимита

Crawl capacity показывает, какую нагрузку сайт способен выдерживать при обращениях поискового робота. Здесь учитываются прежде всего скорость и стабильность ответов сервера. Если сайт нормально отвечает на запросы, Google может увеличивать интенсивность обхода. Если ответы становятся медленными или сервер начинает возвращать ошибки 5xx либо статус 429, частота запросов может быть снижена.

В документации Google этот механизм называется crawl capacity limit, или hostload: это ограничение на объём нагрузки, которую поисковый робот создаёт при обходе сайта. В опубликованных Google материалах 2026 года отдельно подчёркивается, что новый сайт начинает с консервативного значения по умолчанию, а дальнейшее увеличение допустимой нагрузки зависит не только от технического состояния сайта, но и от наличия потребности в более интенсивном обходе. Поэтому возраст домена, размер компании или известность бренда сами по себе не дают сайту повышенного лимита.

Crawl demand отвечает на другой вопрос: насколько поисковой системе вообще необходимо обращаться к конкретным URL. Здесь имеют значение важность и популярность страниц, а также изменения уже известного содержимого. Если страница часто меняется или представляет для поиска заметный интерес, у Google может быть больше причин возвращаться к ней. Если содержимое давно не менялось и не представляет большого интереса, потребность в частом обходе может быть ниже.

Отсюда следует ещё один важный вывод: количество страниц само по себе не определяет частоту обхода сайта. Google отдельно опровергает представление о том, что небольшие сайты автоматически обходятся реже крупных. Имеет значение не сам размер сайта, а то, сколько его содержимого действительно требует обхода и насколько часто оно меняется.

Фактический объём обхода складывается из взаимодействия этих двух факторов. В SEO для этого обычно используют выражение «краулинговый бюджет», хотя в документации Google основной акцент сделан именно на crawl capacity и crawl demand. Это не фиксированное число запросов, которое сайт получает на сутки, а изменяющийся баланс между возможностями инфраструктуры и потребностью поисковой системы в обработке URL.

Ещё одна деталь из свежей редакции документации

Стоит отдельно упомянуть менее очевидное уточнение из той же редакции документации от июля 2026 года: ограничение crawl capacity (пропускной способности обхода) является общим для всех поисковых роботов Google, а не устанавливается отдельно для каждого из них.

Если один из роботов Google, например робот для обхода изображений или робот, связанный с рекламными системами, заметно увеличивает нагрузку на сайт, это учитывается при оценке общей нагрузки. Соответственно, возможности для дальнейшего обхода основным поисковым роботом могут уменьшиться. Речь идёт об одном и том же ресурсе сервера, а не о независимых квотах для разных роботов.

Иными словами, crawl capacity относится к инфраструктуре сайта в целом, а не к персональной квоте одного Googlebot. Растущая активность других роботов, в том числе собирающих данные для систем искусственного интеллекта, тоже может создавать нагрузку на эту инфраструктуру. К этому вопросу книга вернётся отдельно.

Почему это вообще не проблема (для большинства сайтов)

Здесь нужна честная оговорка, которую сам Google регулярно повторяет в своей документации: для большинства сайтов краулинговый бюджет не становится отдельной проблемой. Вопрос прежде всего актуален для крупных сайтов, а также для ресурсов, которые создают большое количество URL автоматически, например за счёт параметров и фильтров. Для сайта на несколько сотен страниц при нормальной технической организации обычно хватает доступной ёмкости, чтобы поисковый робот регулярно обходил его целиком.

Ситуация меняется по мере роста сайта. Именно тогда ограничения, связанные с обходом, начинают иметь практическое значение. Одна и та же техническая недоработка, например фасетная навигация, создающая URL для каждой комбинации фильтров, на небольшом сайте может добавить всего несколько десятков лишних адресов. Google без особых последствий обойдёт их вместе с остальными страницами.

На каталоге с миллионами товаров ситуация уже другая. Та же фасетная навигация способна создать URL-пространство, на порядки превышающее количество действительно значимых страниц. В результате поисковому роботу приходится тратить ресурс обхода на адреса вроде «цвет=синий&размер=42&скидка=да», которые никогда не станут самостоятельными посадочными страницами, вместо того чтобы быстрее добраться до действительно важных URL, например карточки нового товара.

Именно в этом проявляется практический смысл взаимодействия crawl capacity (способности сайта выдерживать нагрузку от поискового робота) и crawl demand (потребности поисковой системы обходить страницы сайта). Google описывает результат этого взаимодействия как набор URL, которые Googlebot может и хочет обходить. «Может» связано с доступной ёмкостью, а «хочет» с потребностью в обходе конкретных страниц.

При этом большое количество URL само по себе не делает их более приоритетными. Если поисковой системе практически не нужны тысячи страниц, созданных комбинациями фильтров, их количество не увеличивает спрос на обход. Но доступные для обхода URL всё равно могут расходовать часть ресурса, который мог бы быть направлен на более важные страницы. Именно поэтому на крупных сайтах управление URL-пространством становится самостоятельной SEO-задачей. обхода.

Что реально расходует бюджет впустую

В технических аудитах чаще всего повторяется один и тот же набор источников избыточного обхода, который не приносит сайту заметной пользы. Это фасетная навигация и параметры сортировки и фильтрации, создающие большое количество URL с почти одинаковым содержимым; внутренний поиск по сайту, страницы которого случайно стали доступными для обхода и индексации; дубли, возникающие из-за нескольких URL одного и того же товара или материала; постраничная навигация в бесконечных лентах, где количество страниц может расти практически без ограничений.

Ни один из этих случаев сам по себе не означает ошибку разработчиков. Каждый механизм может решать вполне понятную пользовательскую задачу. Проблема появляется при большом количестве таких URL. То, что для пользователя выглядит как удобный фильтр или сортировка, для поискового робота становится ещё одним адресом в URL-пространстве сайта. Если таких адресов тысячи или миллионы, они начинают конкурировать за тот же ресурс обхода, который нужен действительно важным страницам.

Экономика масштаба, а не абстрактный лимит

Возвращаясь к контрасту, с которого начиналась эта тема: для сайта на пятьсот страниц вопрос о том, сколько ресурсов Google готов выделить на его обход, обычно не становится узким местом. При таком масштабе доступной ёмкости, как правило, достаточно, и даже неоптимальная архитектура не успевает создать серьёзную конкуренцию между URL.

Для сайта на пять миллионов URL ситуация принципиально другая. Здесь уже важно, какие страницы поисковый робот действительно успевает обходить. Если значительная часть ресурса уходит на параметры, фильтры, дубли и другие малозначимые URL, важные страницы могут дольше оставаться необойденными и, соответственно, не переходить к следующим этапам обработки. В том числе это может проявляться в статусе «Discovered — currently not indexed».

Главный практический вывод этой главы заключается в следующем: краулинговый бюджет — не фиксированный лимит, который нужно однажды выяснить и затем попытаться увеличить. Он складывается из двух факторов: насколько быстро и стабильно сайт способен обрабатывать запросы поискового робота и насколько разумно организовано его URL-пространство.

На первый фактор можно влиять качеством инфраструктуры и обработкой нагрузки, на второй — архитектурой сайта. Сам «бюджет» напрямую настроить нельзя, потому что отдельного переключателя или фиксированной квоты для него не существует.

О том, как избыточное URL-пространство формируется из параметров, фильтров, сортировки и пагинации и как управлять им уже на уровне архитектуры сайта, а не только выявлять последствия, — в следующей главе.

Глава 7. URL-структура и пространство сайта

У одной карточки товара в интернет-магазине может быть не один URL, а десятки и даже сотни. Сортировка по цене, популярности или новизне. Фильтры по цвету, размеру, бренду и наличию. Постраничная навигация для каждой комбинации параметров. Параметр сессии, который CMS добавляет автоматически. Такое множество адресов обычно не создаётся намеренно. Оно становится результатом вполне разумных по отдельности решений, связанных с устройством каталога и удобством интерфейса.

Для пользователя это разнообразие практически незаметно: он выбирает нужные параметры фильтра, сортирует товары и переходит по страницам каталога. Для поисковой системы каждая уникальная комбинация параметров может быть отдельным URL. Если таких адресов становится слишком много, они начинают занимать ресурс обхода, который мог бы использоваться для действительно важных страниц сайта.

Именно поэтому URL-структуру крупного сайта нельзя рассматривать только как вопрос удобства навигации или читаемости адресов. Она определяет объём пространства, которое поисковой системе потенциально приходится обнаруживать и обходить. Чем больше это пространство отличается от набора действительно нужных для поиска страниц, тем выше риск расходовать ресурс обхода на URL, которые не имеют самостоятельной ценности.

Откуда берётся лишнее пространство

Источников избыточного URL-пространства на практике немного, и они повторяются от сайта к сайту. Самый очевидный источник — параметры фильтрации и сортировки. Когда пользователь одновременно применяет несколько фильтров, количество комбинаций быстро растёт. Уже три-четыре параметра с несколькими значениями каждый могут создавать тысячи технически уникальных URL, содержимое которых при этом отличается лишь незначительно.

Параметры сессии и отслеживания тоже исторически были источником большого количества дублей. Речь идёт, например, о метках аналитических систем и идентификаторах сессии. Сегодня эта проблема встречается реже: современные CMS и системы аналитики обычно не требуют добавлять такие параметры к каждому основному URL. Однако полностью исключать их влияние на структуру сайта нельзя.

Ещё один источник — внутренний поиск. Если страницы с результатами поиска доступны для обхода, практически каждый пользовательский запрос может создавать новый URL. Количество таких комбинаций почти не ограничено, а для большинства сайтов эти страницы не представляют самостоятельной ценности для поискового трафика. Но здесь нет универсального правила: некоторые сайты намеренно создают и продвигают страницы результатов внутреннего поиска под конкретные запросы. В таком случае это уже осознанное архитектурное решение, а не техническая ошибка.

Наконец, есть пагинация — разбиение длинных списков на отдельные страницы. Само по себе оно не является проблемой и часто необходимо для нормальной работы каталога или архива. Проблема возникает, когда архитектура позволяет создавать практически бесконечную последовательность страниц, в том числе с комбинациями параметров, которые не имеют самостоятельной ценности.

Что случилось с готовыми инструментами для решения этой задачи

Здесь стоит сделать историческую оговорку, потому что часть рекомендаций, которые до сих пор встречаются в материалах по SEO, уже устарела.

Долгое время в Google Search Console существовал специальный инструмент для работы с параметрами URL. Он появился ещё в 2009 году в Google Webmaster Tools и позволял указать, как поисковой системе следует обращаться с конкретным параметром: учитывать ли его при определении содержимого страницы или считать его несущественным. В марте 2022 года Google объявил о закрытии этого инструмента. Компания объяснила решение тем, что, по её оценке, лишь около 1% настроек действительно помогали управлять обходом. 26 апреля 2022 года инструмент окончательно прекратил работу.

После этого Google сделал ставку на автоматическое определение назначения параметров. Для случаев, когда владельцу сайта необходимо явно ограничить обход, компания по-прежнему указывает на robots. txt как один из доступных механизмов. При этом robots. txt управляет именно обходом URL, а не их индексацией, поэтому использовать его как универсальную замену прежнему инструменту нельзя.

С пагинацией произошла похожая история. Раньше на страницах последовательности рекомендовали использовать атрибуты rel=«next» и rel=«prev», чтобы сообщить поисковой системе о связи между отдельными страницами списка. В марте 2019 года Google сообщил, что больше не использует эти атрибуты как сигнал для индексации. Позднее представитель компании уточнил, что Google фактически перестал учитывать их значительно раньше. При этом сама пагинация никуда не исчезла: страницы последовательности по-прежнему могут быть обычными URL, которые поисковая система обнаруживает и обходит.

Из этой истории стоит вынести не ностальгию по исчезнувшим инструментам, а более практический вывод. Рекомендации, которые когда-то давали владельцу сайта дополнительный способ управлять конкретным механизмом обхода, со временем могут потерять актуальность. Поисковая система совершенствует собственные алгоритмы, а часть задач переносится с ручной настройки на автоматическую обработку.

Поэтому управление URL-пространством сегодня в большей степени начинается с архитектуры самого сайта: какие URL вообще могут появляться, какие связи между ними создаёт навигация и какие из них действительно имеют самостоятельную ценность. Точечные директивы остаются полезными, но они уже не заменяют продуманную структуру сайта.

Три инструмента и путаница вокруг них

Для управления тем, какие URL из разросшегося пространства сайта поисковая система будет обходить и индексировать, используются три разных механизма. Путаница между ними часто приводит к тому, что техническое решение не даёт ожидаемого результата.

Robots. txt

Robots. txt управляет самим обходом URL. Если путь запрещён правилами robots. txt, поисковый робот не должен обращаться к нему напрямую. Это позволяет не расходовать ресурс обхода, о котором шла речь в главе 6.

Но у такого решения есть важное ограничение. Если поисковый робот не может получить страницу, он не может прочитать её содержимое, включая директивы noindex или canonical, если они находятся в HTML. При этом сам URL всё равно может попасть в поисковую базу,

например если на него ведут внешние ссылки. В таком случае Google может показывать адрес без содержимого страницы.

Поэтому robots. txt не стоит использовать как универсальный способ запретить индексацию. Если задача заключается именно в том, чтобы страница не попадала в индекс, блокировка обхода может помешать поисковой системе увидеть соответствующую директиву. Robots. txt прежде всего подходит для управления самим обходом, особенно когда запросы к определённым URL создают ненужную нагрузку на сервер.

Noindex

Noindex решает другую задачу. Эта директива сообщает поисковой системе, что страницу не следует включать в индекс. Она может находиться в HTML страницы или передаваться через HTTP-заголовок. Чтобы увидеть noindex, поисковый робот должен получить документ и обработать его.

Отсюда следует важное практическое правило: robots. txt и noindex нельзя рассматривать как два независимых запрета, которые нужно обязательно использовать одновременно. Если URL закрыт в robots. txt, Google может не получить страницу и, следовательно, не увидеть находящуюся в ней директиву noindex. В результате URL при определённых условиях всё равно может появиться в поиске.

Конец ознакомительного фрагмента.