https://htmlacademy.ru/blog/js/brauzer-google

1. Нажата клавиша «g»

Когда вы нажимаете клавишу «g», браузер получает событие и запускается механизм автоподстановки. В зависимости от алгоритма браузера и его режима (включена ли функция «инкогнито») в выпадающем окне под строкой URL пользователю будет предложено определённое количество вариантов для автоподстановки.

Большинство алгоритмов автоподстановки ранжируют рекомендации в зависимости от истории поиска и оставленных закладках. Некоторые браузеры (например, Rockmelt) даже предлагают профили друзей на Facebook. Когда пользователь планирует напечатать в адресной строке «[google.com](http://google.com/)», ничего из вышеперечисленного не играет роли, но тем не менее выполнится большое количество кода, а рекомендации будут обновляться с каждой новой напечатанной буквой. Возможно, браузер предложит перейти на [google.com](http://google.com/), до того, как пользователь вобьёт адрес целиком.

2. Клавиша «enter» нажата до конца

В качестве некой нулевой точки можно выбрать момент, когда клавиша Enter на клавиатуре нажата до конца и находится в нижнем положении. В этой точке замыкается электрическая цепь этой клавиши и небольшое количество тока отправляется по электросхеме клавиатуры, которая сканирует состояние каждого переключателя клавиши и конвертирует сигнал в целочисленный код клавиши (в данном случае — 13). Затем контроллер клавиатуры конвертирует код клавиши для передачи его компьютеру. Как правило, сейчас передача происходит через USB или Bluetooth, а раньше клавиатура подключалась к компьютеру с помощью коннекторов PS/2 или ADB.

*В случае USB-клавиатуры:*

* Для работы USB-контуру клавиатуры требуется 5 вольт питания, которые поступают через USB-контроллер на компьютере.
* Сгенерированный код клавиши хранится в регистре внутренней памяти клавиатуры, который называется «конечной точкой» (endpoint).
* USB-контроллер компьютера опрашивает эту конечную точку каждые 10 микросекунд и получает хранящийся там код клавиши.
* Затем это значение поступает в USB SIE (Serial Interface Engine) для конвертации в один или более USB-пакетов, которые формируются по низкоуровневому протоколу USB.
* Эти пакеты затем пересылаются с помощью различных электрических сигналов через D+ и D- контакты с максимальной скоростью 1,5 Мб/сек — поскольку HID-устройства (Human Interface Device) всегда были «низкоскоростными».
* Этот последовательный сигнал далее декодируется в USB-контроллере компьютера и интерпретируется универсальным драйвером HID-устройства (клавиатуры). Затем значение кода клавиши передаётся на «железный» уровень абстракции операционной системы.

*В случае виртуальной клавиатуры (тачскрин):*

* Когда пользователь прикладывает палец к современному ёмкостному тач-экрану, небольшое количество тока передаётся к пальцу. Это замыкает цепь через электростатическое поле проводящего слоя и создаёт падение напряжения в этой точке экрана. Экранный контроллер затем инициирует прерывание, сообщающее координату «клика».
* Затем мобильная ОС оповещает текущее открытое приложение о событии клика в одном из GUI-элементов (в этом случае — кнопках виртуальной клавиатуры).
* Виртуальная клавиатура вызывает программное прерывание для отправки сообщения «клавиша нажата» обратно в ОС.
* Это прерывание оповещает текущее открытое приложение о возникновении события «нажатия клавиши».

2.1 Возникло прерывание [не для USB-клавиатур]

Клавиатура отправляет сигналы в свою «линию запросов прерываний» (IRQ), которая затем сопоставляется с «вектором прерывания» (целое число) контроллером прерываний. Процессор использует «таблицу дескрипторов прерываний» (IDT) для сопоставления векторов прерываний с функциями («обработчики прерываний») ядра. Когда появляется прерывание, процессор (CPU) обновляет IDT вектором прерывания и запускает соответствующий обработчик. Таким образом, в дело вступает ядро.

2.2 (На Windows) Сообщение WM\_KEYDOWN отправлено приложению

HID передаёт событие нажатой клавиши драйверу KBDHID.sys, который конвертирует его в [скан-код](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D0%B0%D0%BD-%D0%BA%D0%BE%D0%B4) (scancode). В данном конкретном случае скан-код — VK\_RETURN (0x0D). Драйвер KDBHID.sys связывается с драйвером KBDCLASS.sys (драйвер классов клавиатуры). Он отвечает за безопасную обработку всего ввода с клавиатуры. В дальнейшем этот драйвер вызывает Win32K.sys (после возможной передачи сообщения через установленные сторонние клавиатурные фильтры). Всё это происходит в режиме ядра.

Win32K.sys определяет, какое окно активно в данный момент, с помощью функции GetForegroundWindow(). Этот API обеспечивает обработку окна адресной строки в браузере. Затем главный «насос сообщений» Windows вызывает SendMessage(hWnd, WM\_KEYDOWN, VK\_RETURN, lParam). lParam — это битовая маска, которая указывает на дальнейшую информацию о нажатии клавиши: счётчик повторов (в этом случае 0), актуальный скан-код (может зависеть от OEM, но VK\_RETURN обычно не зависит от этого), информацию о том, были ли нажаты дополнительные клавиши (например, Alt, Shift, Ctrl — в нашем случае не были) и некоторые другие данные.

В API Windows есть функция SendMessage, которая помещает сообщение в очередь для конкретного обработчика окон (hWnd). После этого для обработки всех сообщений очереди вызывается главная функция обработки сообщений (WindowProc), присвоенная обработчику hWnd.

Окно (hWnd), активное в данный момент, представляет из себя контрол обработки и в этом случае у WindowsProc есть обработчик для сообщений WM\_KEYDOWN. Этот код изучает третий параметр, который поступил в SendMessage (wParam) и, поскольку это VK\_RETURN, понимает, что пользователь нажал клавишу ENTER.

2.3 (В OS X) Событие NSEVent KeyDown отправлено приложению

Сигнал прерывания активирует событие прерывания в драйвере I/O Kit клавиатуры. Драйвер переводит сигнал в код клавиатуры, который затем передаётся процессу OS X под названием WindowServer. В результате, Windows Server передаёт событие любому подходящему (активному или «слушающему») приложению через Mach-порт, в котором событие помещается в очередь. Затем события могут быть прочитаны из этой очереди потоками с достаточными привилегиями, чтобы вызывать функцию mach\_ipc\_dispatch. Чаще всего это происходит и обрабатывается с помощью основного цикла NSApplication через NSEvent в NSEventype KeyDown.

2.4 (В GNU/Linux) Сервер Xorg слушает клавиатурные коды

В случае графического X server, для получения нажатия клавиши будет использован общий драйвер событий evdev. Переназначение клавиатурных кодов скан-кодам осуществляется с помощью специальных правил и карт X Server. Когда [маппинг](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%BF%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) скан-кода нажатой клавиши завершён, X server посылает символ в window manager (DWM, metacity, i3), который затем отправляет его в активное окно. Графический API окна, получившего символ, печатает соответствующий символ шрифта в нужном поле.

**3. Парсинг URL**

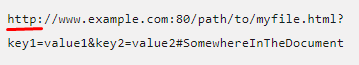
**3.1 Это URL или поисковый запрос?**

Когда пользователь не вводит протокол или доменное имя, то браузер отправляет то, что человек напечатал, поисковой машине, установленной по умолчанию. Часто к URL добавляется специальный текст, который позволяет поисковой машине понять, что информация передана из URL-строки определённого браузера.

**3.2 Что такое URL? Из чего состоит URL?**

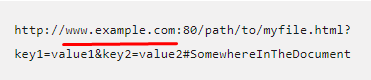
URL - **Uniform Resource Locator -** «единообразный указатель местонахождения ресурса» - Единый локатор ресурсов — адрес ресурса в сети Интернет.  URL это адрес, который выдан уникальному ресурсу в интернете. Сейчас URL позиционируется как часть более общей системы идентификации ресурсов [URI](https://ru.wikipedia.org/wiki/URI), сам термин URL постепенно уступает место более широкому термину [**URI**](https://ru.wikipedia.org/wiki/URI). URL имеет свои стандарты, и регулируется соответсвующей организацией.

Возьмем для примера следующий URL:

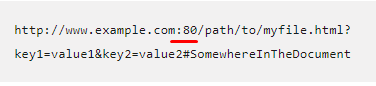


**http** – протокол, схема обращения к ресурсу которому отправляется запрос. Браузер поддерживает много протоколов, некоторые из них:

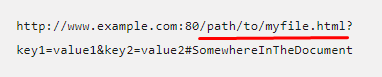
* HTTP, HTTPS (Hyper Text Transfer Protocol) – протоколы передачи гипертекста. Используются при пересылке Web-страниц.
* FTP (File Transfer Protocol) – протокол передачи файлов по сети.
* SMTP (Simple Mail Transfer Protocol) — протокол, который задает набор правил для передачи почты.



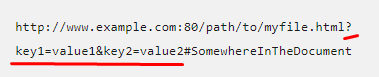
**www.example.com -** доменное имя хоста в системе DNS или тут также может находится IP-адрес хоста, на котором размещена страница.



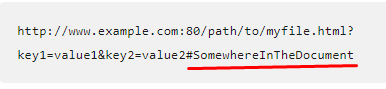
**:80** - порт. Отображает технический параметр, используемый для доступа к ресурсам на веб-сервере. Обычно подразумевается, что веб-сервер использует стандартные порты HTTP-протокола (80 для HTTP и 443 для HTTPS) для доступа к своим ресурсам.



**/path/to/myfile.html** - это адрес ресурса на веб-сервере. Раньше, адрес отображал местоположение реального файла в реальной директории на веб-сервере. В наши дни это чаще всего абстракция, позволяющая обрабатывать адреса и отображать тот или иной контент из баз данных.



**?key1=value1&key2=value2** —Это дополнительные параметры, которые браузер передает веб-серверу (методом GET). Строка начинается с символа -? -. Параметры - список пар ключ/значение, которые разделены символом &. Веб-сервер может использовать эти параметры для исполнения дополнительных команд перед тем, как отдать ресурс. Каждый веб-сервер имеет свои собственные правила обработки этих параметров и узнать их можно, только спросив владельца сервера.



**#SomewhereInTheDocument -** якорь на другую часть того же самого ресурса. Якорь представляет собой вид "закладки" внутри ресурса, которая переадресовывает браузер на "заложенную" часть ресурса. В HTML-документе, например, браузер может переместиться в точку, где установлен якорь; в видео- или аудио-документе браузер может перейти к времени, на которое ссылается якорь. Часть URL после #, которая также известна как идентификатор фрагмента, никогда не посылается на сервер вместе с запросом.

В URL могут также присутствовать и другие составные части.

**3.3 Что делает браузер, если он определил, что введен урл, но не введен протокол?**

До недавнего времени, браузер автоматически подставлял протокол https. Но в последнее время – Google Chrome/Chromium с 2010 г., Safari и Mozilla Firefox с 2012г - браузер проверяет список «предзагруженных HSTS. Это список сайтов, которые требуют, чтобы к ним обращались только по HTTPS.

HTST - HTTP Strict Transport Security - механизм, принудительно активирующий защищённое соединение через протокол HTTPS.

* Если нужный сайт есть в этом списке, то браузер отправляет ему запрос через HTTPS вместо HTTP. В противном случае, начальный запрос посылается по HTTP. При этом сайт может использовать политику HSTS, но не находиться в списке HSTS — в таком случае на первый запрос по HTTP будет отправлен ответ о том, что необходимо отправлять запросы по HTTPS.

**3.4 Конвертация не-ASCII Unicode символов в название хоста**

* Браузер проверяет имя хоста на наличие символов, отличных от a-z, A-Z, 0-9, , или ..
* В случае доменного имени [google.com](http://google.com/) никаких проблем не будет, но если бы домен содержал не-ASCII символы, то браузер бы применил кодировку [Punycode](https://en.wikipedia.org/wiki/Punycode) для этой части URL.

**4. Определение DNS**

* Браузер проверяет наличие домена в своём кэше.
* Если домена там нет, то браузер пытается найти нужный адрес в файле hosts - его расположение отличается в разных ОС.
* Если домен нигде не закэширован и отсутствует в файле hosts, тогда браузер отправляет запрос к сетевому DNS-серверу(dns-resolver). Как правило, это локальный роутер или DNS-сервер Интернет-провайдера (Интернет-провайдер его и предоставляет).
* Если локальный или на стороне провайдера DNS-сервер «не знает» нужный адрес, то запрашивается рекурсивный поиск, который проходит по списку вышестоящих DNS-серверов, пока не будет найдена запись, а затем возвращается результат.

5. Открытие сокета

Когда браузер получает IP-адрес конечного сервера, то он берёт эту информацию и данные об используемом порте из URL (80 порт для HTTP, 443 для HTTPS) и осуществляет вызов функции socket системной библиотеки и запрашивает поток TCP сокета — AF\_INET и SOCK\_STREAM.

* Этот запрос сначала проходит через транспортный уровень, где собирается TCP-сегмент. В заголовок добавляется порт назначения, исходный порт выбирается из динамического пула ядра (ip\_local\_port\_range в Linux).
* Получившийся сегмент отправляется на сетевой уровень, на котором добавляется дополнительный IP-заголовок. Также включаются IP-адрес сервера назначения и адрес текущей машины — после этого пакет сформирован.
* Пакет передаётся на канальный уровень. Добавляется заголовок кадра, включающий MAC-адрес сетевой карты (NIC) компьютера, а также MAC-адрес шлюза (локального роутера). Как и на предыдущих этапах, если ядру ничего не известно о MAC-адресе шлюза, то для его нахождения отправляется широковещательный ARP-запрос.

На этой точке пакет готов к передаче через:

* [Ethernet](http://en.wikipedia.org/wiki/IEEE_802.3)
* [WiFi](https://en.wikipedia.org/wiki/IEEE_802.11)
* [По сотовой связи](https://en.wikipedia.org/wiki/Cellular_data_communication_protocol)

В случае интернет-соединения большинства частных пользователей или небольших компаний пакет будет отправлен с компьютера, через локальную сеть, а затем через модем (MOdulator/DEModulator), который транслирует цифровые единицы и нули в аналоговый сигнал, подходящий для передачи по телефонной линии, кабелю или беспроводным телефонным соединениям. На другой стороне соединения расположен другой модем, который конвертирует аналоговый сигнал в цифровые данные и передаёт их следующему [сетевому узлу](https://en.wikipedia.org/wiki/Computer_network#Network_nodes), где происходит дальнейший анализ данных об отправителе и получателе.

В конечном итоге пакет доберётся до маршрутизатора, управляющего локальной подсетью. Затем он продолжит путешествовать от одного роутера к другому, пока не доберётся до сервера назначения. Каждый маршрутизатор на пути будет извлекать адрес назначения из IP-заголовка и отправлять пакет на следующий хоп. Значение поля TTL (time to live) в IP-заголовке будет каждый раз уменьшаться после прохождения каждого роутера. Если значение поля TTL достигнет нуля, пакет будет отброшен (это произойдёт также если у маршрутизатора не будет места в текущей очереди — например, из-за перегрузки сети).

Во время TCP-соединения происходит множество подобных запросов и ответов.

5.1 Жизненный цикл TCP-соединения

a. Клиент выбирает номер начальной последовательности (ISN) и отправляет пакет серверу с установленным битом SYN для открытия соединения.

b. Сервер получает пакет с битом SYN и, если готов к установлению соединения, то:

* Выбирает собственный номер начальной последовательности;
* Устанавливает SYN-бит, чтобы сообщить о выборе начальной последовательности;
* Копирует ISN клиента +1 в поле ACK и добавляет ACK-флаг для обозначения подтверждения получения первого пакета.

c. Клиент подтверждает соединение путём отправки пакета:

* Увеличивает номер своей начальной последовательности;
* Увеличивает номер подтверждения получения;
* Устанавливает поле ACK.

d. Данные передаются следующим образом:

* Когда одна сторона отправляет N байтов, то увеличивает значение поля SEQ на это число.
* Когда вторая сторона подтверждает получение этого пакета (или цепочки пакетов), она отправляет пакет ACK, в котором значение поля ACK равняется последней полученной последовательности.

e. Закрытие соединения:

* Сторона, которая хочет закрыть соединение, отправляет пакет FIN;
* Другая сторона подтверждает FIN (с помощью ACK) и отправляет собственный FIN-пакет;
* Инициатор прекращения соединения подтверждает получение FIN отправкой собственного ACK.

6. TLS handshake

* Клиентский компьютер отправляет сообщение ClientHello серверу со своей версией протокола [TLS](https://ru.wikipedia.org/wiki/TLS), списком всех поддерживаемых алгоритмов шифрования и методов компрессии данных.
* Сервер отвечает клиенту сообщением ServerHello, и используя ассиметричное шифрования передает сообщение содержащее версию TLS, выбранный метод шифрования, выбранные методы компрессии и публичный сертификат сервиса, подписанный центром сертификации. Сертификат содержит публичный ключ, который будет использоваться клиентом для шифрования оставшейся части процедуры «рукопожатия» (handshake), пока не будет согласован симметричный ключ.
* Клиент подтверждает сертификат сервера с помощью своего списка центров сертификации. Если сертификат подписан центром из списка, то серверу можно доверять, и клиент генерирует строку псевдослучайных байтов и шифрует её с помощью публичного ключа сервера. Эти случайные байты могут быть использованы для определения симметричного ключа.
* Сервер расшифровывает случайные байты с помощью своего секретного ключа и использует эти байты для генерации своей копии симметричного мастер-ключа.
* Клиент отправляет серверу сообщение Finished, шифруя хеш передачи с помощью симметричного ключа.
* Сервер генерирует собственный хеш, а затем расшифровывает полученный от клиента хеш, чтобы проверить, совпадёт ли он с собственным. Если совпадение обнаружено, сервер отправляет клиенту собственный ответ Finished, также зашифрованный симметричным ключом.
* После этого TLS-сессия передаёт данные приложения (HTTP), зашифрованные с помощью подтверждённого симметричного ключа.

7. Протокол HTTP

Если используемый браузер был создан Google, то вместо отправки HTTP-запроса для получения страницы, он отправит запрос, чтобы попытаться «договориться» с сервером об «апгрейде» протокола с HTTP до [SPDY](https://ru.wikipedia.org/wiki/SPDY) («спиди»).

Если клиент использует HTTP-протокол и не поддерживает SPDY, то отправляет серверу запрос следующей формы:

GET / HTTP/1.1

Host: google.com

Connection: close

[другие заголовки]

Где [другие заголовки] — это серия пар «ключ: значение», разбитых переносом строки. (Здесь предполагается, что в использованном браузере нет никаких ошибок, нарушающих спецификацию HTTP. Также предполагается, что браузер использует HTTP/1.1, в противном случае он может не включать заголовок Host в запрос и версия, отданная в ответ на GET-запрос может быть HTTP/1.0 или HTTP/0.9).

HTTP/1.1 определяет опцию закрытия соединения («close») для отправителя — с её помощью происходит уведомление о закрытии соединения после завершения ответа. К примеру:

Connection: close

Приложения HTTP/1.1, которые не поддерживают постоянные соединения, обязаны включать опцию «close» в каждое сообщение.

После отправки запроса и заголовков, браузер отправляет серверу единичную пустую строку, сигнализируя о том, что содержимое сообщения закончилось.

Сервер отвечает специальным кодом, который обозначает статус запроса и включает ответ следующей формы:

200 OK

[заголовки ответа]

После этого посылается пустая строка, а затем оставшийся контент HTML-страницы [www.google.com](http://www.google.com/). Сервер может затем закрыть соединение, или, если того требуют отправленные клиентом заголовки, сохранять соединение открытым для его использования следующими запросами.

Если HTTP-заголовки отправленные веб-браузером включают информацию, которой серверу достаточно для определения версии файла, закэшированного в браузере и этот файл не менялся со времени последнего запроса, то ответ может принять следующую форму:

304 Not Modified

[заголовки ответа]

И, соответственно, клиенту не посылается никакого контента, вместо этого браузер «достаёт» HTML из кэша.

После разбора HTML, браузер (и сервер) повторяет процесс загрузки для каждого ресурса (изображения, стили, скрипты, favicon.ico и так далее), на который ссылается HTML-страница, но при этом изменяется адрес каждого запроса c GET / HTTP/1.1 на GET /$(относительный URL ресурса www.google.com) HTTP/1.1.

Если HTML ссылается на ресурс, размещённый на домене, отличном от [google.com](http://google.com/), то браузер возвращается к шагам, включающим разрешение доменного имени, а затем заново проходит процесс до текущего состояния, но уже для другого домена. Заголовок Host в запросе вместо [google.com](http://google.com/) будет установлен на нужное доменное имя.

7.1 Обработка HTTP-запросов на сервере

HTTPD (HTTP Daemon) является одним из инструментов обработки запросов/ответов на стороне сервера. Наиболее популярные HTTPD-серверы это Apache или Nginx для Linux и IIS для Windows.

1. HTTPD (HTTP Daemon) получает запрос.
2. Сервер разбирает запрос по следующим параметрам:
   * Метод HTTP-запроса (GET, POST, HEAD, PUT или DELETE). В случае URL-адреса, который пользователь напечатал в строке браузера, мы имеем дело с GET-запросом.
   * Домен. В нашем случае — [google.com](http://google.com/).
   * Запрашиваемые пути/страницы, в нашем случае — / (нет запрошенных путей, / — это путь по умолчанию).
3. Сервер проверяет существование виртуального хоста, который соответствует [google.com](http://google.com/).
4. Сервер проверяет, что [google.com](http://google.com/) может принимать GET-запросы.
5. Сервер проверяет, имеет ли клиент право использовать этот метод (на основе IP-адреса, аутентификации и прочее).
6. Если на сервере установлен модуль перезаписи (mod\_rewrite для Apache или URL Rewrite для IIS), то он сопоставляет запрос с одним из сконфигурированных правил. Если находится совпадающее правило, то сервер использует его, чтобы переписать запрос.
7. Сервер находит контент, который соответствует запросу, в нашем случае он изучит индексный файл.
8. Далее сервер разбирает («парсит») файл с помощью обработчика. Если Google работает на PHP, то сервер использует PHP для интерпретации индексного файла и направляет результат клиенту.

8. За кулисами браузера

Задача браузера заключается в том, чтобы показывать пользователю выбранные им веб-ресурсы, запрашивая их с сервера и отображая в окне просмотра. Как правило такими ресурсами являются HTML-документы, но это может быть и PDF, изображения или контент другого типа. Расположение ресурсов определяется с помощью URL.

Способ, который браузер использует для интерпретации и отображения HTML-файлов описан в спецификациях HTML и CSS. Эти документы разработаны и поддерживаются консорциумом W3C (World Wide Web Consortium), которая занимается стандартизацией веба.

Интерфейсы браузеров сильно похожи между собой. У них есть большое количество одинаковых элементов:

* Адресная строка, куда вставляются URL-адреса;
* Кнопки возврата на предыдущую и следующую страницу;
* Возможность создания закладок;
* Кнопки обновления страницы (рефреш) и остановки загрузки текущих документов;
* Кнопка «домой», возвращающая пользователя на домашнюю страницу.

Высокоуровневая структура браузера

Браузер включает следующие компоненты:

* Пользовательский интерфейс: В него входит адресная строка, кнопки продвижения вперёд/назад, меню закладок и так далее. Сюда относятся все элементы, кроме окна, в котором собственно отображается веб-страница.
* «Движок» браузера: Распределяет действия между движком рендеринга и интерфейсом пользователя.
* «Движок» рендеринга: Отвечает за отображение запрашиваемого контента. К примеру, если запрашивается HTML, то «движок» разбирает код HTML и CSS, а затем отображает полученный контент на экране.
* Сетевая часть: с помощью сетевых функций браузер обрабатывает вызовы, вроде HTTP-запросов, с применением различных реализаций для разных платформ.
* Бэкенд интерфейса (UI): Используется для отрисовки базовых виджетов, вроде комбо-боксов и окон.
* Интерпретатор JavaScript: Используется для парсинга и выполнения JavaScript-кода.
* Хранилище данных: Браузеру может понадобиться локально хранить некоторые данные (например, cookie). Кроме того, браузеры поддерживают различные механизмы хранения, такие как localStorage, IndexedDB, WebSQL и FileSystem.

9. Парсинг HTML

Движок рендеринга начинает получать содержимое запрашиваемого документа от сетевого механизма браузера. Как правило, контент поступает кусками по 8Кб. Главной задачей HTML-парсера является разбор разметки в специальное дерево.

Получающееся на выходе дерево («parse tree») — это дерево DOM-элементов и узлов атрибутов. DOM — сокращение от Document Object Model. Это модель объектного представления HTML-документа и интерфейс для взаимодействия HTML-элементов с «внешним миром» (например, JavaScript-кодом). Корнем дерева является объект «Документ».

Алгоритм разбора

HTML-нельзя «распарсить» с помощью обычных анализаторов (нисходящих или восходящих). Тому есть несколько причин:

* Прощающая почти что угодно природа языка.
* Тот факт, что браузеры обладают известной толерантностью к ошибкам и поддерживают популярные ошибки в HTML.
* Процесс парсинга может заходить в тупик. В других языках код, который требуется разобрать, не меняется в процессе анализа, в то время как в HTML с помощью динамического кода (например, скриптовые элементы, содержащие вызовы document.write()) могут добавляться дополнительные токены, в результате чего сам процесс парсинга модифицирует вывод.

Невозможность использования привычных технологий парсинга приводит к тому, что разработчики браузеров реализуют собственные механизмы разбора HTML. Алгоритм парсинга подробно описан в [спецификации HTML5](http://www.w3.org/TR/html5/).

Алгоритм состоит из двух этапов: токенизации и создания дерева.

Действия после завершения парсинга

После этого браузер начинает подгружать внешние ресурсы, связанные со страницей (стили, изображения, скрипты и так далее).

На этом этапе браузер помечает документ, как интерактивный и начинает разбирать скрипты, находящиеся в «отложенном» состоянии: то есть те из них, что должны быть исполнены после парсинга. После этого статус документа устанавливается в состояние «complete» и инициируется событие загрузки («load»).

Важный момент: ошибки «Invalid Syntax» при разборе не может быть, поскольку браузеры исправляют любой «невалидный» контент и продолжают работу.

10. Интерпретация CSS

* Во время разбора браузер парсит CSS-файлы, содержимое тегов <style> и атрибутов «style» c помощью «[лексической и синтаксической грамматики CSS](http://www.w3.org/TR/CSS2/grammar.html)».
* Каждый CSS-файл разбирается в объект StyleSheet, каждый из таких объектов содержит правила CSS с селекторами и объектами в соответствии с грамматикой CSS.
* Парсер CSS может быть как восходящим, так и нисходящим.

11. Рендеринг страниц

* Путём перебора DOM-узлов и вычисления для каждого узла значений CSS-стилей создаётся «Дерево рендера» (Render Tree или Frame Tree).
* Вычисляется предпочтительная ширина каждого узла в нижней части дерева — для этого суммируются значения предпочтительной ширины дочерних узлов, а также горизонтальные поля, границы и отступы узлов.
* Вычисляется реальная ширина каждого узла сверху-вниз (доступная ширина каждого узла выделяется его потомкам).
* Вычисляется высота каждого узла снизу-вверх — для этого применяется перенос текста и суммируются значения полей, высоты, отступов и границ потомков.
* Вычисляются координаты каждого узла (с использованием ранее полученной информации).
* Если элементы плавающие или спозиционированы абсолютно или относительно, предпринимаются более сложные действия. Более подробно они описаны [здесь](http://dev.w3.org/csswg/css2/) и [здесь](http://www.w3.org/Style/CSS/current-work).
* Создаются слои для описания того, какие части страницы можно анимировать без необходимости повторного растрирования. Каждый объект (фрейма или рендера) присваивается слою.
* Для каждого слоя на странице выделяются текстуры.
* Объекты (рендеры/фреймы) каждого слоя перебираются и для соответствующих слоёв выполняются команды отрисовки. Растрирование может осуществляться процессором или возможна отрисовка на графическом процессоре (GPU) через D2D/SkiaGL.
* Все вышеперечисленные шаги могут требовать повторного использования значений, сохранённых с последнего рендеринга страницы, такая инкрементальная работа требует меньше затрат.
* Слои страницы отправляются процессу-компоновщику, где они комбинируются со слоями для другого видимого контента (интерфейс браузера, iframe-элементы, addon-панели).
* Вычисляются финальные позиции слоёв и через Direct3D/OpenGL отдаются композитные команды. Командные буферы GPU освобождаются для асинхронного рендеринга и фрейм отправляется для отображения на экран.

12. Рендеринг GPU

* Во время процесса рендеринга уровни графических вычислений могут использовать процессор компьютера или графический процессор (GPU).
* Во втором случае уровни графического программного обеспечения делят задачу на множество частей, что позволяет использовать параллелизм GPU для вычисления плавающей точки, которое требуется для процесса рендеринга.

13. Вызванное пользователем и пост-рендеринговое исполнение

После завершения рендеринга, браузер исполняет JavaScript-код в результате срабатывания некоего часового механизма (так работают дудлы на странице Google) или в результате действий пользователя (ввод поискового запроса в строку и получение рекомендаций в ответ). Также могут срабатывать плагины вроде Flash или Java (но не в рассматриваемом примере с домашней страницей Google). Скрипты могут потребовать обработки дополнительных сетевых запросов, изменять страницу или её шаблон, что приведёт к следующему этапу рендеринга и отрисовки.