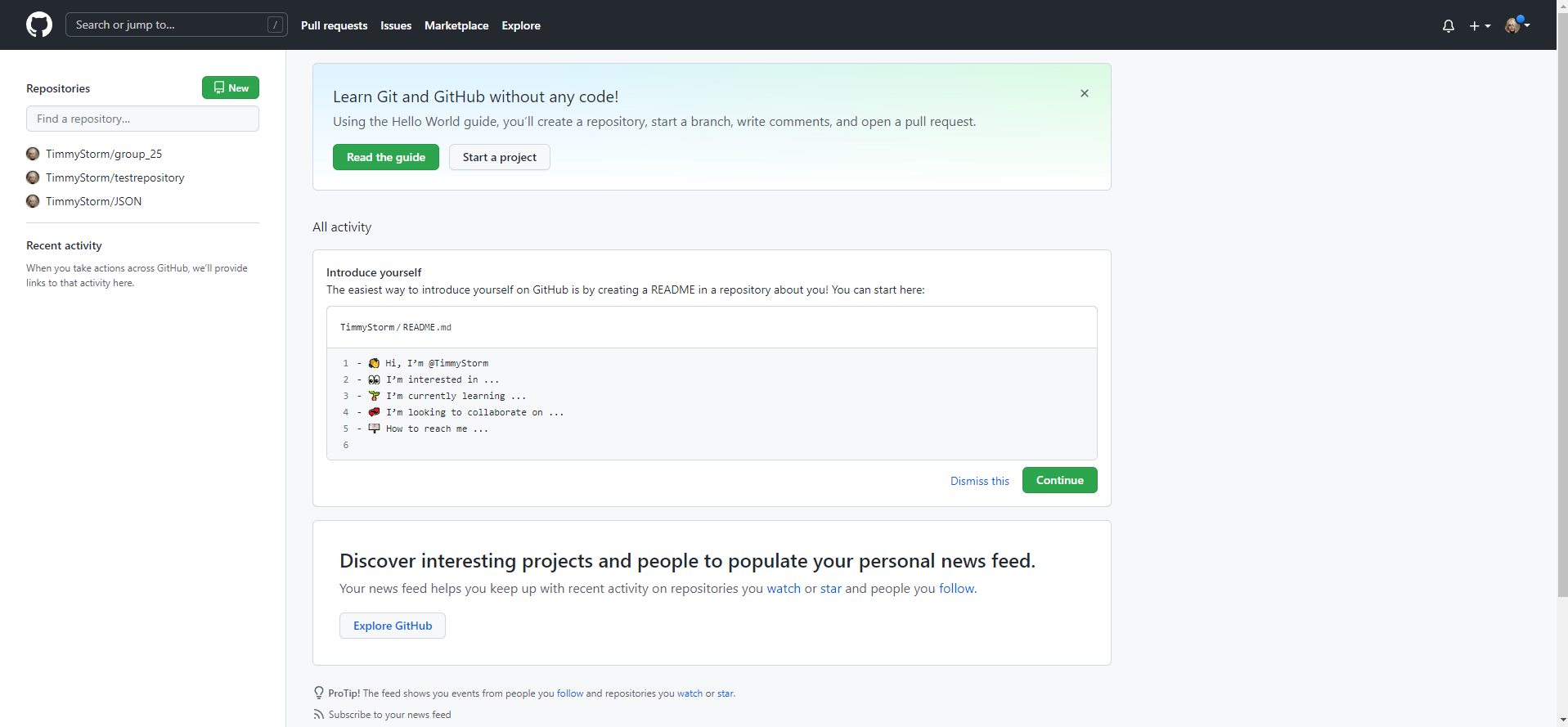
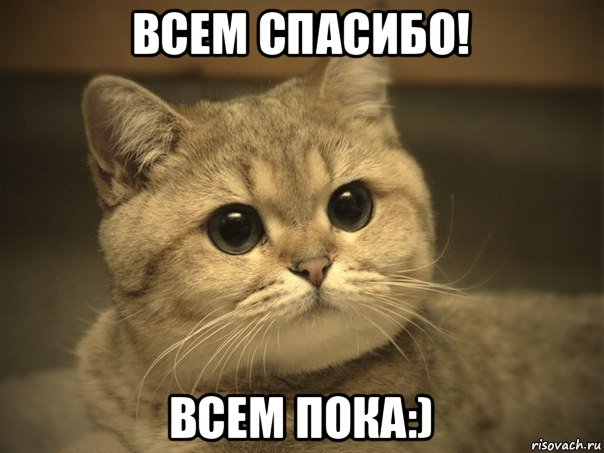
**Что происходит, когда пользователь набирает в браузере адрес сайта- github.com?**

Отображается страница github спустя три секунды 😊

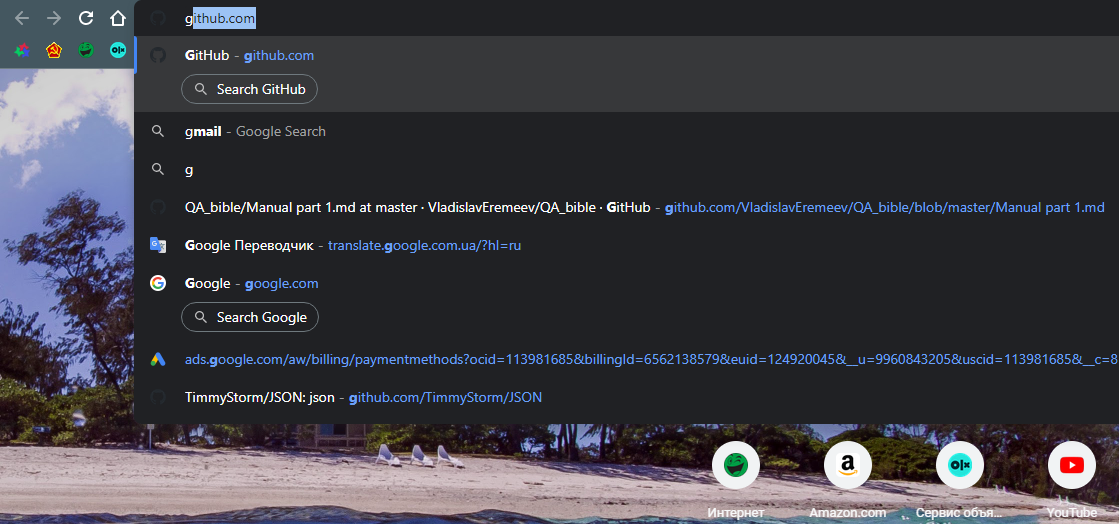




Задача браузера заключается в том, чтобы показывать пользователю выбранные им веб-ресурсы, запрашивая их с сервера и отображая в окне просмотра. Как правило такими ресурсами являются HTML-документы, но это может быть и PDF, изображения или контент другого типа. Расположение ресурсов определяется с помощью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

Механизм устроен следующим образом:

**1. Набираем первую букву “g”. Дальше браузер включает механизм автоподстановки для нашего запроса.**



Но прежде чем продвинуться дальше, давайте разберём разницу между- сервером, веб-сервером, сервисом, микросервисом и сайтом.

**С точки зрения "железа", «веб-сервер»** — это компьютер, на который установлен web-server. Хранит файлы сайта (HTML-документы, CSS-стили, JavaScript-файлы, картинки и другие) и доставляет их на устройство конечного пользователя (веб-браузер и т.д.). Он подключён к сети Интернет и может быть доступен через доменное имя, подобное mozilla.org.

**С точки зрения ПО, веб-сервер включает в себя** несколько компонентов, которые контролируют доступ веб-пользователей к размещённым на сервере файлам, как минимум — это HTTP-сервер. HTTP-сервер — это часть ПО, которая понимает URL-адреса (веб-адреса) и HTTP (протокол, который ваш браузер использует для просмотра веб-страниц). Основная задача веб-сервера - отображать содержимое веб-сайта.

**Web-сервис(служба)** – программа, которая организовывает взаимодействие между сайтами. Информация с одного портала передается на другой. Самые известные способы реализации веб-сервисов: REST и SOAP.

**Архитектурный стиль микросервисов** -это программная архитектура, которая может использоваться наряду с веб-сервисами. Этот метод включает в себя разделение больших программных приложений на отдельные модули, где микросервисы выполняют уникальные процессы и взаимодействуют через APIs. Архитектура, в которой все компоненты системы объединены в отдельные компоненты, которые могут быть построены, развернуты и масштабированы индивидуально.

**Сайт-** документ или информационный ресурс Всемирной паутины, доступ к которому осуществляется с помощью веб-браузера.

Веб-сервис не имеет пользовательского интерфейса. Веб-сайт имеет пользовательский интерфейс или графический интерфейс.

Веб-сервисы предназначены для взаимодействия других приложений через Интернет. Веб-сайты предназначены для использования людьми.

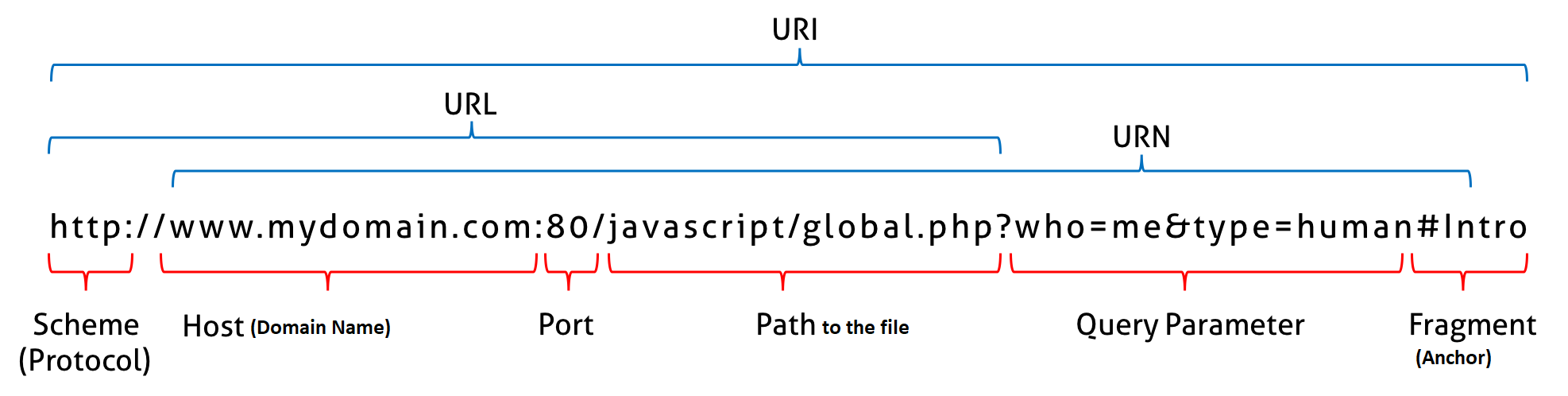
Веб-сервисы не зависят от платформы, так как используют открытые протоколы. Веб-сайты являются кроссплатформенными, так как требуют настройки для работы в разных браузерах, операционных системах и т. д.

Доступ к веб-сервисам осуществляется с помощью HTTP-методов - GET, POST, PUT, DELETE и т. д. Доступ к веб-сайтам осуществляется с помощью компонентов GUI - кнопок, текстовых полей, форм и т. д.

**2. После нажатия enter- Браузер определяет это url или поисковый запрос. В нашем случае github.com- это url.**

**Url(Uniform Resource Locator)- указатель размещения сайта в Интернете.**

Рассмотрим подробнее из чего он состоит:



Сейчас существует серьезная тенденция на сокращение длины адресов сайтов. Именно поэтому правилом хорошего тона для новых ресурсов стало использование имени домена без префикса WWW.

Теперь у браузера есть следующая информация об URL:

Protocol «HTTP»

Использовать «Hyper Text Transfer Protocol»

Resource «/»

Показать главную (индексную) страницу

**Браузер проверяет список HSTS (HTTP Strict Transport Security).** Это список сайтов, которые требуют, чтобы к ним обращались только по HTTPS. Механизм принудительно активирующий защищённое соединение через протокол https. Если браузер есть в этом списке- то браузер отправит запрос по протоколу https, если нет- то по протоколу http.

**Далее Браузер проверяет имя хоста на наличие символов**, отличных от a-z, A-Z, 0-9, -, или . Если бы домен содержал не-ASCII символы, то браузер бы применил кодировку Punycode для этой части URL.

Каждому URL-адресу в Интернете назначен уникальный IP-адрес. IP-адрес принадлежит компьютеру, на котором размещен сервер веб-сайта, к которому мы запрашиваем доступ. Например, github.com имеет IP-адрес 140.82.121.4. Так что, если хотите, вы можете перейти на github.com, набрав https://140.82.121.4 в своем браузере. Всё, что идёт после «https://», и есть домен. В нашем примере домен — github.com = https:// 140.82.121.4

**Domain Name System** – это система иерархической связи, призванная организовать хранение и обмен данными про доменные имена между серверами. DNS - это список URL-адресов и их IP-адресов.

**Из чего состоит доменное имя**

Любой домен состоит из уровней. Уровни домена — это части, которые разделяют точки. Они нужны, чтобы систематизировать работу доменных имён. Считывание доменов идет справа налево.

**Первый уровень** **домена(верхний уровень)-** это часть имени, которая находится справа от последней точки. При регистрации нельзя придумать свой домен первого уровня. Придётся выбирать из существующих.

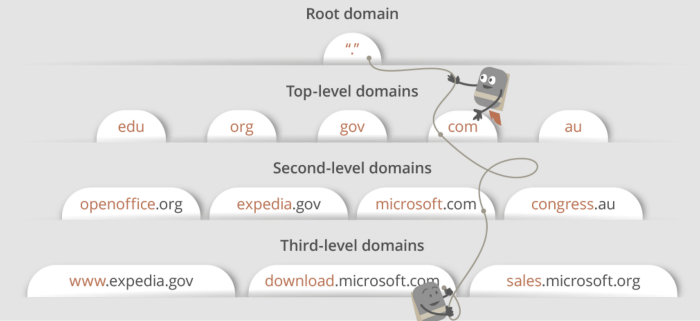
Полный список доменов первого уровня хранится на сайте Управления по присвоению номеров в интернете. Вот некоторые из них: .com, .net, .org, .biz, .info, .ua, .ru, .me.

**Второй уровень домена(основной или материнский)**- это часть имени, которая находится слева от последней точки. Hostiq — это домен второго уровня в имени hostiq.ua.

**Третий уровень домена(субдомен или поддомен )-** это часть имени, которая находится слева от предпоследней точки. Субдомены используют, когда хотят присвоить уникальный адрес разным разделам на сайте.

Субдомены сделают структуру сайта понятнее. Допустим, у вашего сайта есть две версии: русскоязычная и англоязычная. Каждой из них можно присвоить свой субдомен: ru.site.com и en.site.com. Тогда пользователям будет проще сразу перейти на нужную версию сайта.





**DNS-сервер** — это специальный компьютер, который хранит IP-адреса сайтов. Сервер DNS обеспечивает хранение и администрирование данных в иерархической структуре. В его задачи входит передача сведений о доменном имени. Каждый узел имеет свою зону ответственности. Вышестоящий сервер в древовидной структуре предает запрос нижестоящему или другому узлу. Для увеличения скорости загрузки некоторые сервера DNS могут кэшировать данные других узлов.

Настройки каждого домена в интернете хранятся в текстовых файлах на DNS-серверах.

**Что такое DNS-зоны?**

С определенным доменным именем может быть связано несколько ресурсов — сайт и почтовый сервер. У этих ресурсов вполне могут быть разные IP-адреса, что дает возможность повысить надежность и эффективность работы сайта или почтовой системы. Есть у сайтов и поддомены, IP-адреса которых тоже могут быть разными.

Вся эта информация о связи сайта, поддоменов, почтовой системы хранится в специальном файле на DNS-сервере. Его содержимое называется DNS-зона.

**Чтобы найти запись DNS, браузер сначала проверяет четыре кеша.**

● Сначала он проверяет кеш браузера. Браузер поддерживает репозиторий DNS-записей в течение фиксированного периода времени для веб-сайтов, которые вы ранее посещали. Итак, это первое место для запуска DNS-запроса.

● Во-вторых, браузер проверяет кеш ОС основного компьютера для извлечения записи, поскольку ОС также поддерживает кеш записей DNS.

● В-третьих, он проверяет кэш маршрутизатора. Если его нет на вашем компьютере, браузер будет связываться с маршрутизатором, который поддерживает свой собственный кеш записей DNS.

● В-четвертых, он проверяет кеш-память интернет-провайдера. Если все шаги не удастся, браузер перейдет к провайдеру. Ваш интернет-провайдер поддерживает свой собственный DNS-сервер, который включает в себя кеш DNS-записей, которые браузер будет проверять с последней надеждой найти запрошенный вами URL.

**DNS-кэш**

Локальные DNS-серверы умеют кэшировать настройки, нужно это для того, чтобы при повторном посещении страницы загружались быстрее. Из-за этого случаются ситуации, когда владелец домена поменял настройки, но браузер показывает старую страницу, потому что получает IP-адрес из кэша.

**DNS кэш** — это временная база данных, которая содержит записи обо всех последних посещениях и попытках посещений веб-сайтов и другие IP-адреса сайтов.

**3. Если запрошенный URL-адрес отсутствует в кеше, DNS-сервер интернет-провайдера инициирует DNS-запрос, чтобы найти IP-адрес сервера, на котором размещается github.com.**

Не обнаружив подходящих записей в кэше, браузер формирует запрос к DNS-серверам, расположенным в интернете. Браузер может выполнять DNS-запрос по протоколу UDP и TCP. Традиционно запросы и ответы отправляются в виде одной UDP-датаграммы. TCP используется, когда размер данных ответа превышает 512 байт.

**Цель DNS-запроса** - поиск нескольких DNS-серверов в Интернете до тех пор, пока не будет найден правильный IP-адрес для веб-сайта. Этот тип поиска называется рекурсивным поиском, поскольку поиск будет неоднократно продолжаться с DNS-сервера на DNS-сервер, пока он либо не найдет нужный нам IP-адрес, либо не вернет ответ об ошибке, в котором говорится, что он не смог его найти.

**И происходит это следующий образом**- если локально ничего не нашлось сервер идёт на сетевой DNS сервер- это или роутер или DNS сервер провайдера, если у того нет информации- он идёт к корневому серверу, если у него нет такого домена, он обращается к DNS серверу ответственному за зону к которому привязан домен, в нашем случае- это com, но в нём так же может не быть этой инфы, если и он не знает он стучится на NS сервер( в нём google прописывает эту инфу у себя). Этот процесс называется рекурсией.

Рекурсия – это поиск информации по всей системе с последующей выдачей данных на запрос и промежуточным кэшированием ответов от всех опрошенных серверов.

Нерекурсивный поиск выдает только ссылки на сервера, чтобы браузер смог самостоятельно опросить каждый узел.

# **4. Браузер инициирует TCP-соединение с сервером.**

Далее устанавливается TCP соединение с определённым портом, для ТСР есть важная характеристика- это порт по которому нужно установить соединение. Для http- 80 port, https- 443 port. Мы стучимся на 443 port.

Порт- применяется для идентификации программы или процесса для обмена пакетами данных в рамках одного ip адреса.

Для передачи пакетов данных между вашим компьютером (клиентом) и сервером важно установить TCP-соединение. Это соединение устанавливается с помощью процесса, называемого- Трехстороннее рукопожатие TCP / IP. Это трехэтапный процесс, в котором клиент и сервер обмениваются сообщениями SYN (синхронизация) и ACK (подтверждение) для установления соединения.

1. Клиентская машина отправляет SYN-пакет на сервер через Интернет, спрашивая, открыт ли он для новых подключений.

2. Если на сервере есть открытые порты, которые могут принимать и инициировать новые соединения, он ответит подтверждением пакета SYN, используя пакет SYN / ACK.

3. Клиент получит пакет SYN / ACK от сервера и подтвердит его, отправив пакет ACK.

Затем устанавливается TCP-соединение для передачи данных!

**Но так как мы используем https соединение, нам ещё нужно установить защищённое соединение**. Https- это HyperText Transfer Protocol Secure, приставка s(secure)- обозначает использование TLS(transport layer security — Протокол защиты транспортного уровня) или SSL(Secure Sockets Layer — уровень защищённых сокетов). Эти два протокола обеспечивают шифрование. В наше время больше используется TLS, так как он более продвинутый.

В HTTPS-соединении участвуют две стороны: клиент и сервер. Цель рукопожатия SSL/TLS — выполнить всю криптографическую работу для установки безопасного соединения, в том числе проверить подлинность используемого SSL-сертификата и сгенерировать ключ шифрования.

TLS handshake происходит следующим способом:

* Клиентский компьютер отправляет сообщение ClientHello серверу со своей версией протокола [TLS](https://ru.wikipedia.org/wiki/TLS), списком поддерживаемых алгоритмов шифрования и методов компрессии данных.
* Сервер отвечает клиенту сообщением ServerHello, содержащим 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.
* Дальше клиент и сервер- отправляют друг- другу сообщения уже с помощью этого симметричного ключа.

Для каждого соединения с сайтом создается новый секретный ключ.

Но хэш и кэш- это разные вещи.

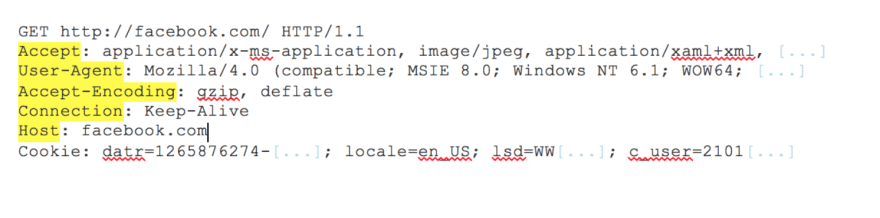
Кэш- это буфер между браузером и интернетом, в котором сохраняются посещённые пользователем страницы на пк, для того что-бы они быстрее загружались.

Хэширование представляет собой преобразование любого объема информации в уникальный набор символов, который присущ только этому массиву входящей информации.

# **5. Браузер отправляет HTTP-запрос на веб-сервер.**

Как только TCP-соединение установлено, пора начинать передачу данных! Браузер отправит запрос GET с запросом на веб-страницу github.com.  Этот запрос также будет содержать дополнительную информацию, такую ​​как идентификация браузера ( заголовок User-Agent ), типы запросов, которые он будет принимать ( заголовок Accept ), и заголовки соединения с просьбой поддерживать соединение TCP для дополнительных запросов. Он также будет передавать информацию, взятую из файлов cookie, которые браузер хранит для этого домена.

Пример запроса GET:



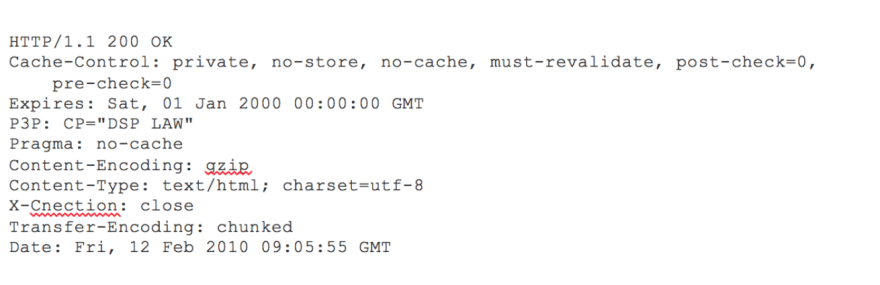
**6. Сервер обрабатывает запрос и отправляет ответ.**

Сервер содержит веб-сервер (например, Apache, IIS, lighttpd, Jagsaw), который получает запрос от браузера и передает его обработчику запросов для чтения и генерации ответа. Обработчик запросов - это программа (написанная на ASP.NET, PHP, Ruby и т. Д.), Которая считывает запрос, его заголовки и файлы cookie для проверки того, что запрашивается, а также при необходимости обновляет информацию на сервере. Затем он соберет ответ в определенном формате (JSON, XML, HTML).

**7. Сервер отправляет ответ HTTP.**

Ответ сервера содержит запрошенную вами веб-страницу, а также код состояния, тип сжатия ( Content-Encoding) , способ кэширования страницы ( Cache-Control ), любые файлы cookie, которые необходимо установить, информацию о конфиденциальности и т. Д.

Пример ответа HTTP-сервера:



Если вы посмотрите на ответ выше, в первой строке будет показан код состояния. Это очень важно, так как сообщает нам статус ответа. Существует пять типов статусов, детализированных с помощью числового кода.

● 1xx обозначает только информационное сообщение.

● 2xx означает какой-то успех.

● 3xx перенаправляет клиента на другой URL.

● 4xx указывает на ошибку со стороны клиента.

● 5xx указывает на ошибку со стороны сервера.

**8. Браузер отображает содержимое HTML.**

Браузер отображает содержимое HTML поэтапно. Во-первых, он визуализирует голый скелет HTML. Затем он проверит теги HTML и отправит запросы GET для дополнительных элементов на веб-странице, таких как изображения, таблицы стилей CSS, файлы JavaScript и т. д. Эти статические файлы кэшируются браузером, поэтому ему не нужно их извлекать снова при следующем посещении страницы. В конце концов, вы увидите, что в вашем браузере появится отрисованная страница- github.com.

Но давайте разберём этот процесс более детально.

**Браузер состоит из следующих компонентов:**

**1.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.

**2.«Движок» браузера:** Распределяет действия между движком рендеринга и интерфейсом пользователя.

**3.«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.

**4.Сетевая часть:** с помощью сетевых функций браузер обрабатывает вызовы, вроде HTTP-запросов, с применением различных реализаций для разных платформ.

**5. Исполнительная часть пользовательского интерфейса:** Используется для отрисовки базовых виджетов, вроде комбо-боксов и окон.

**6.Интерпретатор JavaScript:** Используется для парсинга и выполнения JavaScript-кода.

**7.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.

Браузеру предстоит пройти много шагов, прежде чем HTML-ответ от сервера будет преобразован в пиксели на экране.

Как только браузер получил ответ сервера, он начинает парсить полученную информацию. Этот процесс необходим для преобразования данных в деревья DOM(Document Object Model) и CCOM(CSS Object Model), на основании которых рендерный движок затем создаст изображение сайта на экране.

**Последовательность этих шагов, необходимая для первого отображения страницы, называется «Критический путь рендеринга» (Critical Rendering Path).**

**Существует 6 этапов CRP:**

**1.построение DOM-дерева,**

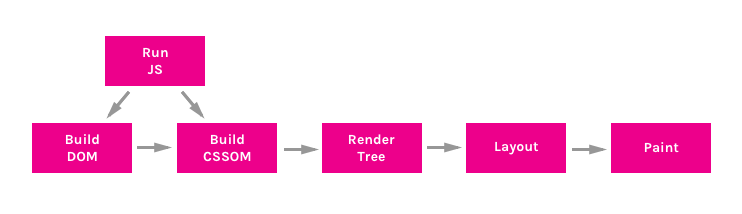
**2.построение CSSOM-дерева,**

**3.запуск JavaScript,**

**4.создание Render-дерева,**

**5.генерация раскладки,**

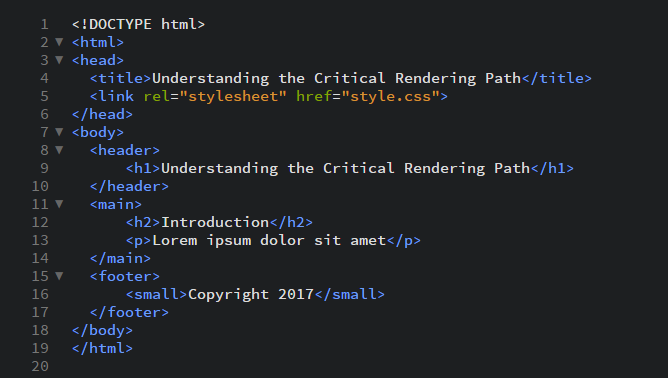
**6.отрисовка.**



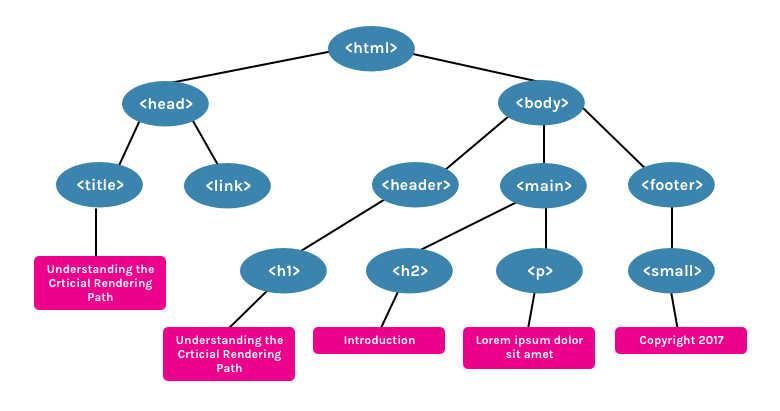
**Построение DOM-дерева**

DOM (объектная модель документа) дерево это объект, представляющий полностью разобранную HTML-страницу. Начиная с корневого элемента <html>, узлы создаются для каждого элемента/текста на странице. Элементы, вложенные в другие элементы, представлены в виде дочерних узлов, и каждый узел содержит полный набор атрибутов для этого элемента. Например, элемент <a> будет иметь атрибут href, связанный с узлом.

Возьмём для примера такой документ:



Из него будет построено такое DOM-дерево:

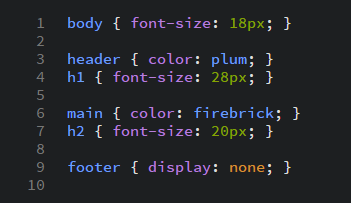


Хорошая новость, касательно HTML, заключается в том, что он может быть исполнен по частям. Документ не должен быть загружен полностью для того, чтобы контент начал появляться на странице. Однако, другие ресурсы, такие как CSS и JavaScript, могут блокировать отрисовку страницы.

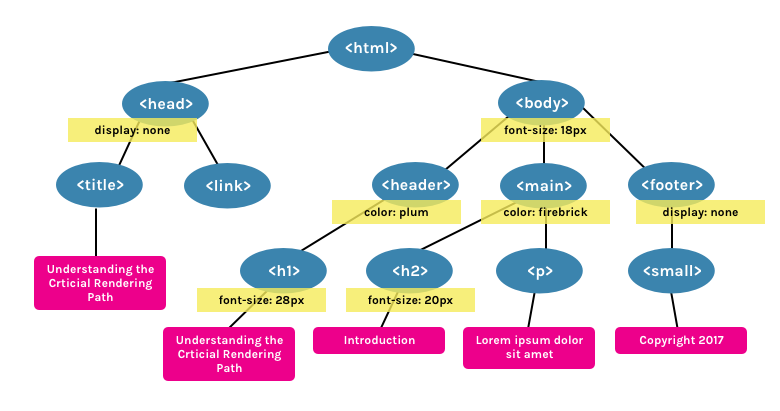
**Построение CSSOM-дерева**

Для css он такой же, как и выше, браузеру необходимо преобразовать этот файл в CSSOM. CSSOM (объектная модель CSS) — это объект, представляющий стили, связанные с DOM.

В файле style.css, подключающемся в ранее упомянутом документе, мы имеем следующий набор стилей:



С его помощью получится следующее CSSOM-дерево:



CSS является одним из наиболее важных элементов критического пути рендеринга, поскольку браузер блокирует показ страницы до тех пор, пока не получит и не обработает все файлы css на вашей странице, CSS блокирует рендеринг.

## Запуск JavaScript

## Когда парсер доходит до тега <script> (не важно внутренний он или внешний), он останавливается, забирает файл (если он внешний) и запускает его. Вот почему, если мы имеем JavaScript-файл, который ссылается на элементы документа, мы обязательно должны поместить его после их появления.

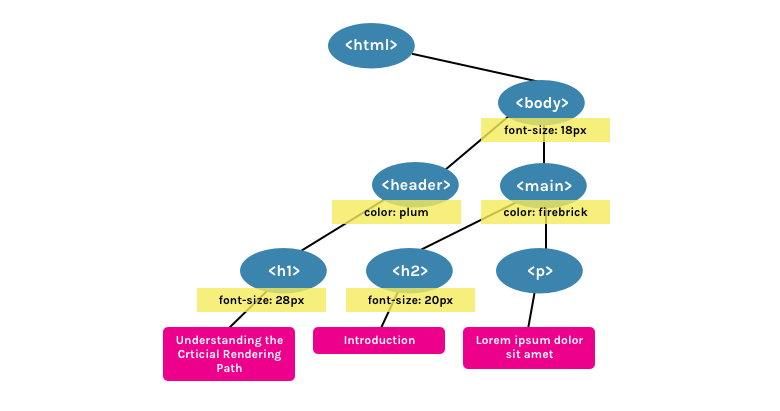
## JavaScript можно загружать асинхронно, указав атрибут async, для того, чтобы избежать блокировки парсера.



**Создание Render-дерева**

После завершения двух процессов синтаксического анализа и определения DOM и CSSOM браузер создает дерево рендера. Дерево рендера - это другое дерево, хранящееся внутри браузера и используемое для представления визуальных элементов. Элементы внутри <head> игнорируются, а также элементы, определенные с display: none. Однако стоит помнить, что элементы со свойством visibility: hidden по-прежнему являются частью дерева.

**На примерах DOM и CSSOM, представленных выше, будет построено такое Render-дерево:**



**Компоновка (Layout )**

Layout – это расчет размера и положения объекта.

В тот момент, когда дерево рендера (render tree) построено, становится возможным этап компоновки (layout). Компоновка зависит от размеров экрана. Этот этап определяет, где и как на странице будут спозиционированы элементы и каковы связи между элементами.

**Отрисовка(Paint/ Repaint)**

Пространство, где отображается сформированная структура называется холст (canvas).

Следующий этап после Layout называется Paint. Размера и положения элементов не достаточно для того чтобы отобразить страницу, нужно знать каким образом «покрасить» эти элементы. На этапе Paint браузер обходит Layout Tree и создает записи о том как будут отрисованы элементы на странице(позиция x,y, ширина, высота, цвет).

Последний этап в нашем списке - отрисовка (paint) пикселей на экране. Когда дерево рендера (render tree) создано, компоновка (layout) произошла, пиксели могут быть отрисованы. На шаге отрисовки, видимый контент страницы может быть преобразован в пиксели, чтобы появиться на экране.

Время, которое займет этот этап, зависит как от величины DOM, так и от того, какие стили применяются. Некоторые стили требуют больше усилий, чтобы быть применёнными, чем другие. Например, сложное градиентное фоновое изображение потребует больше времени, чем простой сплошной цвет на фоне.

