Браузер

Что происходит, когда в адресной строке вы набираете yandex.ru и нажимаете Enter?

1. Парсинг URL: протокол (`http`, `https`) и ресурс (основная часть)
2. В браузер встроен список *HTTP**S***-only доменов, именуемый *HSTS (HTTP Strict Transport Security)*. Если домен в списке найден — подставляется протокол: `https://`. Если нет — `http://`.
3. Получение IP-адреса для запрашиваемого домена, используя DNS
    1. Поиск в DNS кеше браузера.
    2. Если домен не найден — браузер просит операционную систему найти эту запись.
    3. Операционная система ищет в локальном файле `hosts`.
    4. Операционная система ищет в своём DNS-кеше.
    5. Операционная система запрашивает информацию у DNS-сервера, указанного в подключении к сети.
    6. DNS-сервер ищет домен у себя и если не находит — рассылает запросы к известным ему другим DNS-серверам.
4. Браузер открывает подключение, используя порт 80 для *HTTP* и 443 порт для *HTTP**S***
5. Для *HTTP**S*** производится *TLS (Transport Layer Security) Handshake*, где клиент и сервер договариваются об ключе шифрования.

6. Браузер отправляет `GET` запрос, который состоит
    1. Метода: `GET`. Основные методы
        1. `GET` — получить [запись]
        2. `POST` — создать [запись]
        3. `PUT` — сохранить [изменения всей записи]
        4. `PATCH` — дополнить [запись частью новых данных]
        5. `DELETE` — удалить [запись]
    2. URL без домена
    3. Версии *HTTP*
    4. Host: yandex.ru
    5. Connection: close — информация о том, что после возвращения результата, не нужно держать соединение открытым.
    6. Заголовки, в том числе Cookies
    7. Пример:

```
GET / HTTP/1.1 Host: google.com Connection: close [other headers]
```

Plain Text

7. *HTTPD (HTTP Daemon)* на сервере бекенда получает запрос.
8. Сервер парсит запрос *(напоминаю, все передаваемые данные по сети — это простой текст)* на составные части и вызывает свой обработчик

9. Мы не знаем, как именно устроен обработчик этого запроса в Яндексе, но знаем, что в результате вернётся HTML-документ. Ответ состоит
    1. Версии HTTP
    2. Кода ответа. Список популярных.
        1. `200` OK — успешно
        2. `301` Moved Permanently — документ перемещён навсегда. В заголовке `Location` будет новый адрес. Браузер самостоятельно сделает запрос туда.
        3. `304` Not Modified — контент не изменялся. Браузер возьмёт контент из кеша.
        4. `400` Bad Request — запрос сформирован некорректно. Чаще всего ошибка в полях отправляемой формы
        5. `401` Unauthorized — запрос не авторизован. Обычно пользователь не авторизовался.
        6. `403` Forbidden — запрещено. Обычно у пользователя не хватает прав.
        7. `404` Not Found — не найдено документа или обработчика этого URL.
        8. `500` Internal Server Error — внутренняя ошибка сервера. Например, сервер упал с ошибкой при попытке ответа.
        9. `503` Service Unavailable — сервис недоступен. Обычно возвращается, когда сервер перегружен обработкой других запросов.
    3. Заголовков ответа (здесь так же могут прийти изменения для Cookies)
    4. Пример ответа:

```
HTTP/1.1 200 OK Content-Type: text/html; charset=UTF-8 Content-
Length: 17073 [body]
```

Plain Text

10. Браузер начинает парсить тело ответа от сервера кусками по 8 килобайт. Он понимает, что это HTML, глядя на заголовок `Content-Type: text/html`. Если заголовка не будет, браузер попытается угадать тип на основе контента.

11. Для каждого загруженного куска HTML браузер:
    1. Парсит содержимое в древовидную структуру.
    2. Добавляет эту структуру в общий для документа *DOM (Document Object Model)*
    3. Подгружает другие ресурсы, найденные в этом куске: JavaScript, CSS, картинки. Поскольку все принимаемые данные — простой текст — они должны быть обработаны браузером.
        1. Парсинг CSS
        2. Применение CSS
        3. Парсинг JavaScript
        4. Исполнение JavaScript
    4. Принимает решение: можно ли рендерить частично-загруженный документ в текущем состоянии. Например, если сервер отдаёт HTML медленно — браузер будет рендерить страницу кусками.
        1. JavaScript исполняется последовательно, в том порядке, как указан в документе.
        2. Незагруженный JavaScript будет блокировать рендеринг, если тега `<script>` нет атрибута `defer` или `async`
            1. `async` — скрипт не блокирует рендеринг и исполняется сразу после загрузки. Может вызвать проблемы, когда библиотека загрузится позднее, чем код, её использующий.
            2. `defer` — то же самое, что `async`, но с соблюдением порядка в документе. Можно сказать, что всегда нужно использовать этот атрибут.
        3. Незагруженный CSS так же будет блокировать рендеринг куска, но не обязательно до окончания загрузки. Браузер может принять решение рендерить кусок и без него. Если скрипт потом загрузится — перерендерит.
    5. Рендеринг:
        1. Для каждого элемента DOM, браузер производит вычисления его высоты, ширины и позиции на экране, основываясь на свойствах DOM-элемента (в том числе CSS). Этот процесс называется `layout`.

2. Каждый раз, когда изменяются значения стилей, не изменяющие размер элемента — происходит `repaint`, т.е. перерисовка конкретно этого элемента и его потомков.
3. Каждый раз, когда изменяются размеры элемента — происходит `reflow` (перерисовка части страницы) или `relayout` (полная перерисовка страницы), в зависимости от .
4. Отлаживать это можно через *Chrome Developer Tools* во вкладке *Performance*. https://gist.github.com/paulirish/5d52fb081b3570c81e3a
5. Уменьшить количество `reflow` можно выделив анимируемые элементы в отдельный слой: `transform: translate3d(0,0,0)`