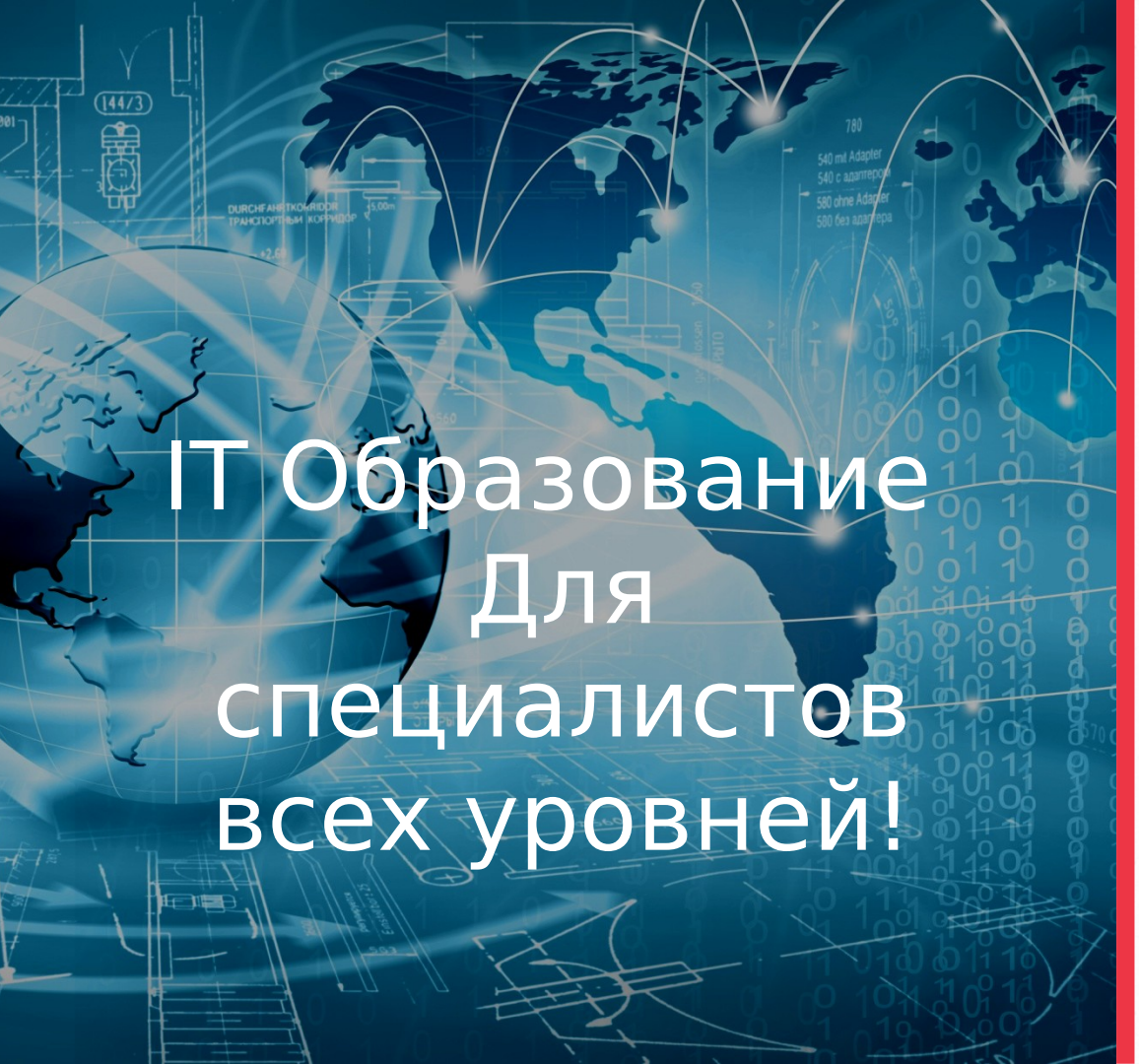


IT Образование Для специалистов всех уровней!

Education
Academy

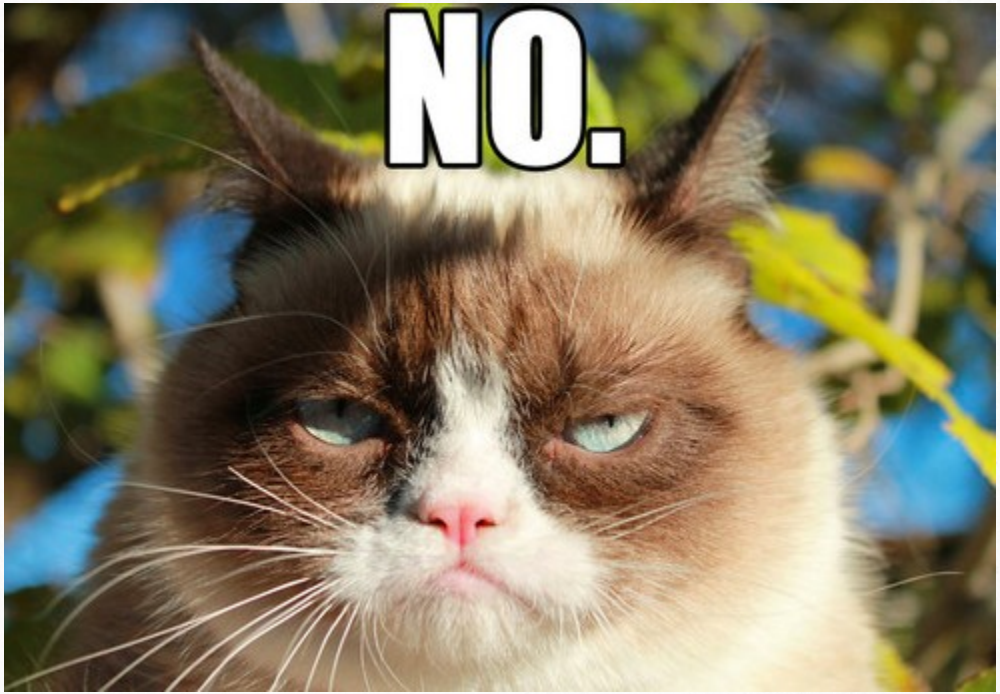
WWW.ITEDUCATE.COM.UA

JavaScript Pro

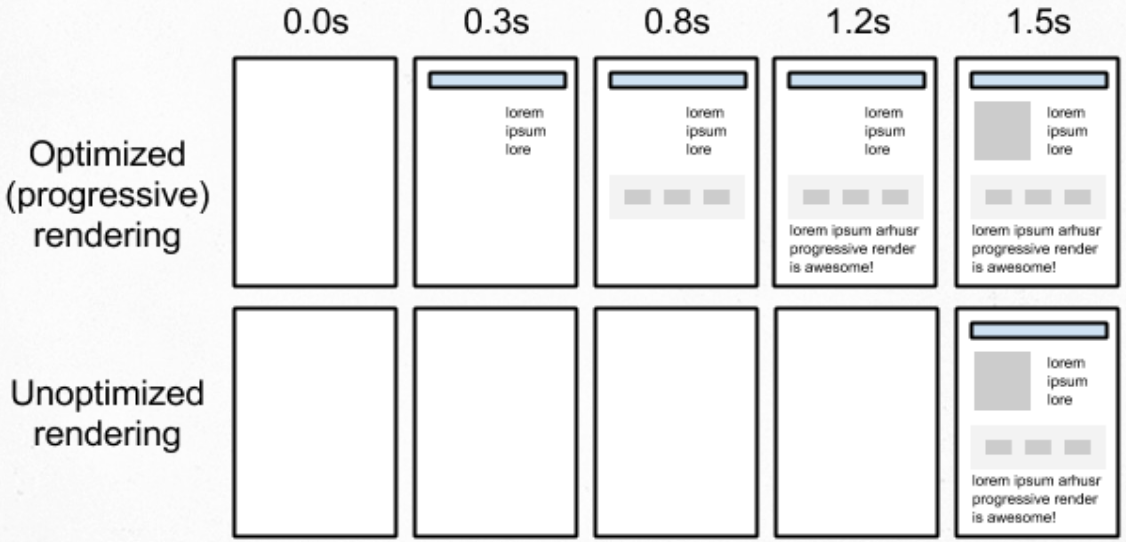
Critical Rendering Path

Оптимизация контента

Slow page load



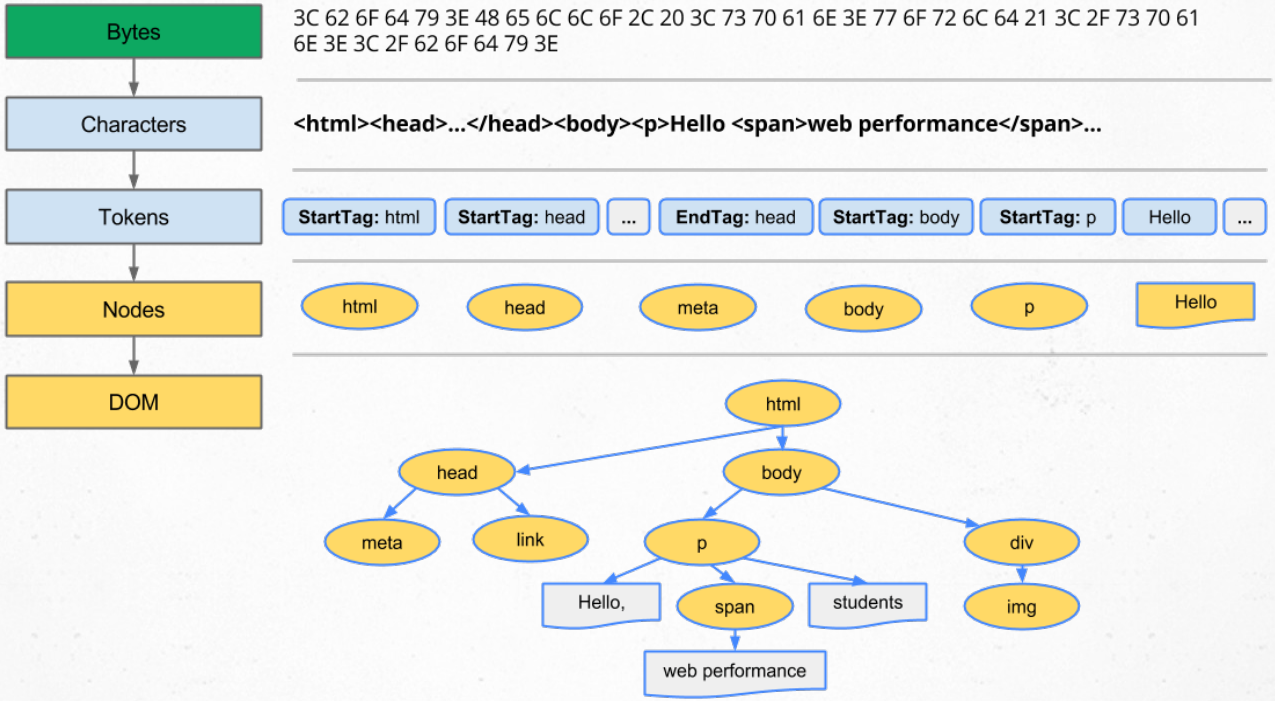
Page render example



Bytes → characters → tokens → nodes → object model.

- Байты → символы → разметка → узлы → объектная модель.
- Браузер преобразует HTML-разметку в объектную модель документа (DOM), а CSS-разметку - в объектную модель таблицы стилей (CSSOM).
- Модели DOM и CSSOM не зависят друг от друга.
- Инструменты разработчика в Chrome позволяют записать и проанализировать этапы создания DOM и CSSOM.

Страница с текстом без форматирования и одним изображением



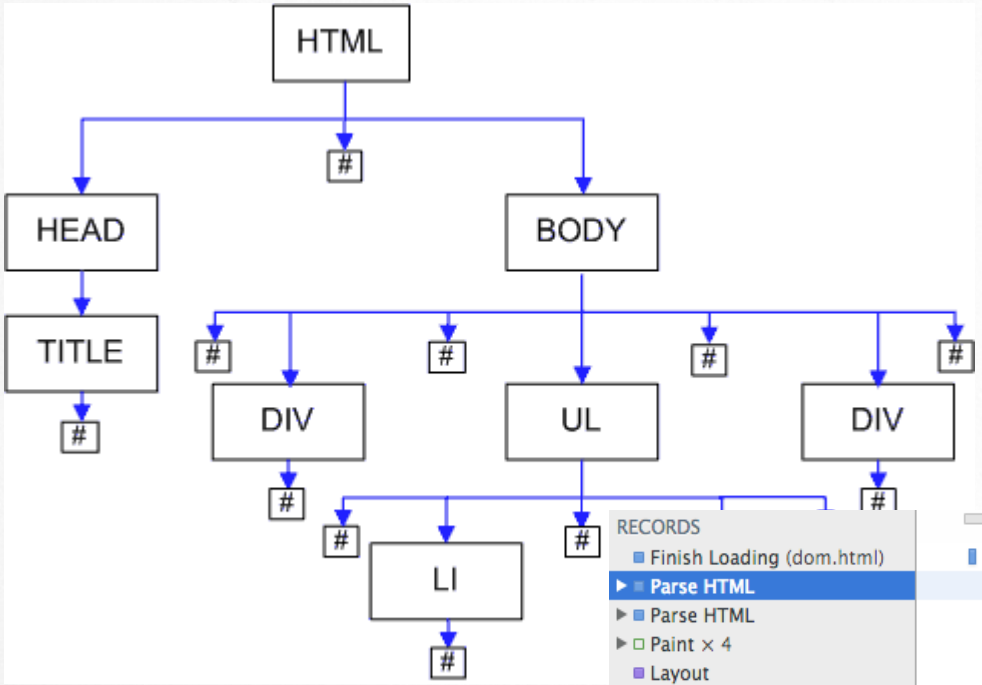
Страница с текстом без форматирования и одним изображением

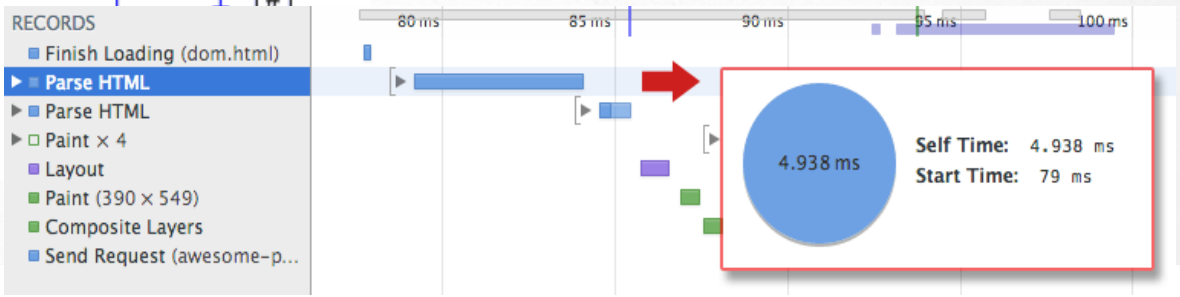
- **Преобразование.** Браузер преобразует байты из HTML-файла, размещенного на диске или в сети, в символы, основываясь на приведенной в файле кодировке (например, UTF-8).
- **Разметка.** На основании стандарта W3C HTML5 браузер выделяет среди символов теги в угловых скобках, такие как <html>, <body> и другие. У каждого тега есть свое значение и свой набор правил.
- **Создание объектов.** С помощью HTML-тегов браузер выделяет в документе объекты с определенными свойствами.
- **Формирование DOM.** Объекты образуют древовидную структуру, повторяющую иерархию HTML-файла, в котором одни теги помещаются в другие. Так, объект p помещается под body, а объект body, в свою очередь, под html, и так далее.

Страница с текстом без форматирования и одним изображением

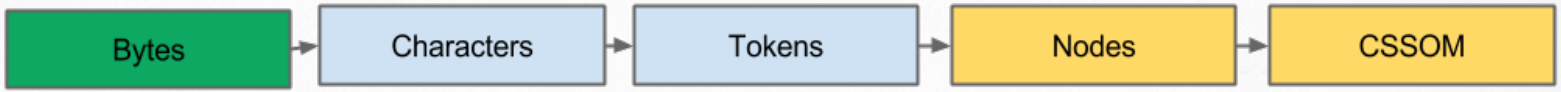
- **Преобразование.** Браузер преобразует байты из HTML-файла, размещенного на диске или в сети, в символы, основываясь на приведенной в файле кодировке (например, UTF-8).
- **Разметка.** На основании стандарта W3C HTML5 браузер выделяет среди символов теги в угловых скобках, такие как <html>, <body> и другие. У каждого тега есть свое значение и свой набор правил.
- **Создание объектов.** С помощью HTML-тегов браузер выделяет в документе объекты с определенными свойствами.
- **Формирование DOM.** Объекты образуют древовидную структуру, повторяющую иерархию HTML-файла, в котором одни теги помещаются в другие. Так, объект p помещается под body, а объект body, в свою очередь, под html, и так далее.

DOM

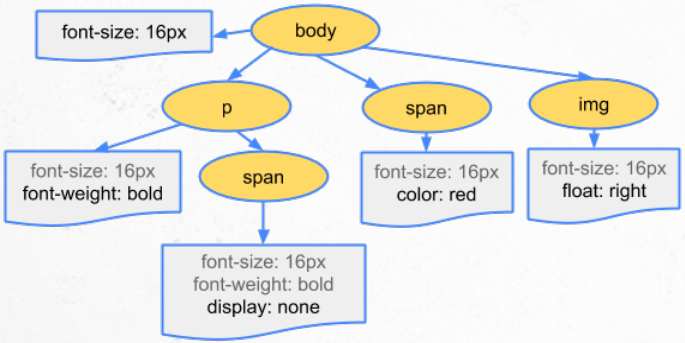

HTML
HEAD
BODY
TITLE
DIV
UL
DIV
LI
#

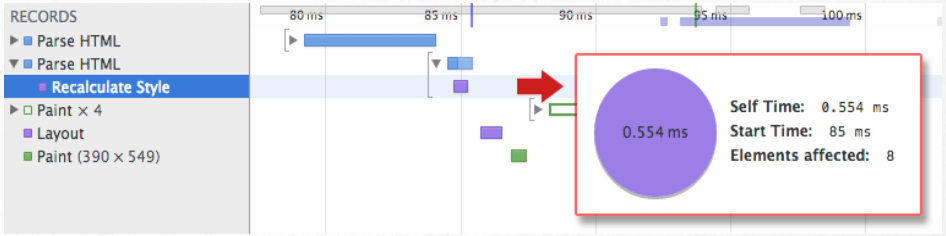

RECORDS
Finish Loading (dom.html)
Parse HTML
Parse HTML
Paint × 4
Layout
Paint (390 × 549)
Composite Layers
Send Request (awesome-p...
80 ms
85 ms
90 ms
95 ms
100 ms
4.938 ms
Self Time: 4.938 ms
Start Time: 79 ms

Модель CSSOM



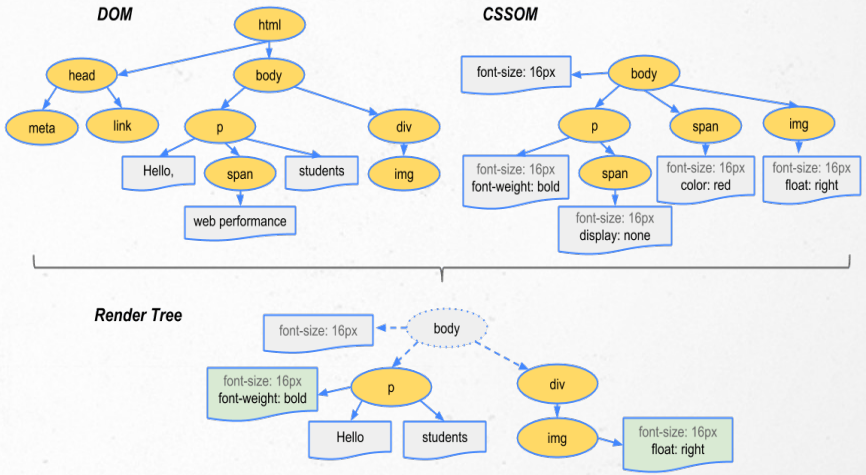
Байты из CSS-файла преобразуются в символы, символы - в теги, а теги - в объекты, которые образуют модель CSSOM:





Браузер объединяет DOM и CSSOM, формируя модель визуализации.

- ***Браузер объединяет DOM и CSSOM, формируя модель визуализации.***
- Модель визуализации содержит только те объекты, которые нужны для вывода страницы на экран.
- Далее формируется макет, отражающий расположение и размер каждого объекта.
- Объекты выводятся на экран.



Браузер объединяет DOM и CSSOM, формируя модель визуализации.

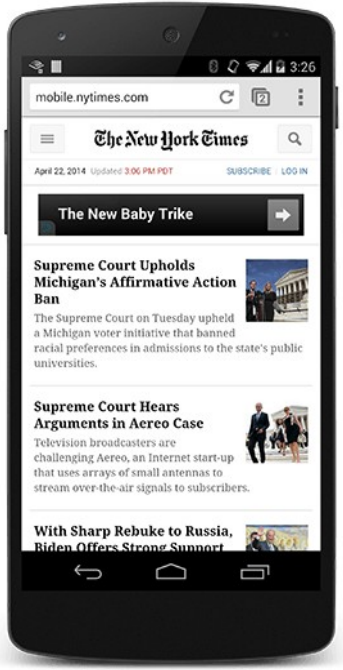
- ***Начиная с основания модели DOM, находит все видимые объекты.***
  - Этот этап не затрагивает элементы, которые не будут видны на странице, например теги скриптов, метатеги и т. п.
  - Он также не затрагивает объекты, помеченные как невидимые с помощью CSS (span отсутствует в модели визуализации, потому что ему присвоен параметр display: none).
- Находит в CSSOM наборы стилей и присваивает их соответствующим объектам.
- Формирует модель из видимых объектов, их содержания и стилей.

Что делает браузер при загрузке?

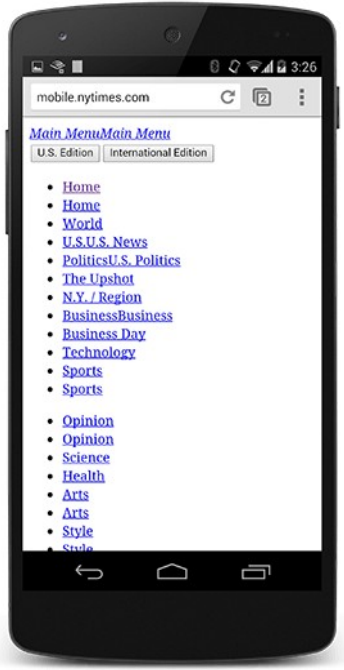
- Обработка HTML-разметки и создание модели DOM.
- Обработка CSS-файла и создание модели CSSOM.
- Создание модели визуализации из DOM и CSSOM.
- Определение формы и расположения объектов, создание макета. (Layout)
- Вывод объектов на экран. (Paint)

Чтобы избежать такой ситуации, браузер обрабатывает CSS-файл сразу же, как только находит его в HTML-документе. В результате страница выводится на экран только после того, как браузер создаст обе модели: и DOM, и CSSOM.

"Нью-Йорк таймс" с CSS



"Нью-Йорк таймс" без CSS



Чтобы избежать такой ситуации, браузер обрабатывает CSS-файл сразу же, как только находит его в HTML-документе. В результате страница выводится на экран только после того, как браузер создаст обе модели: и DOM, и CSSOM.

```
<link href="style.css"    rel="stylesheet">
<link href="style.css"    rel="stylesheet" media="all">
<link href="portrait.css" rel="stylesheet" media="orientation:portrait">
<link href="print.css"    rel="stylesheet" media="print">
```

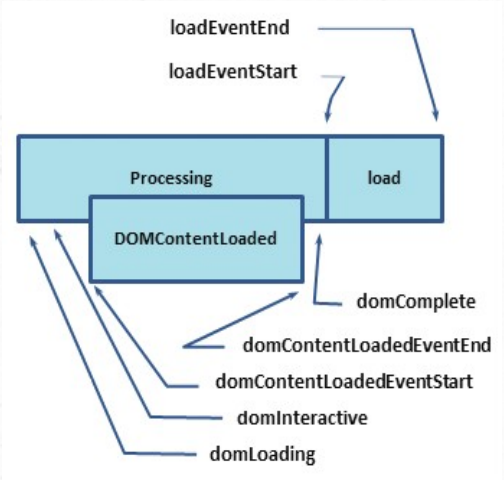
- Первый CSS-файл заблокирует вывод страницы, так как его необходимо обработать.
- Несмотря на медиазапрос, CSS-файл во второй строке также заблокирует вывод страницы. Дело в типе устройства all - браузер присваивает это значение по умолчанию всем CSS-файлам без медиазапроса. Поэтому вторая ссылка в примере эквивалентна первой.
- Третья ссылка содержит медиазапрос с условием, которое будет проверено при загрузке страницы. При портретной ориентации устройства браузер заблокирует вывод страницы и начнет обработку CSS; при альбомной ориентации браузер продолжит создание DOM.
- Четвертый CSS-файл используется только для печати, поэтому браузер не будет блокировать вывод страницы.

Оптимизация JavaScript для быстрой визуализации страницы

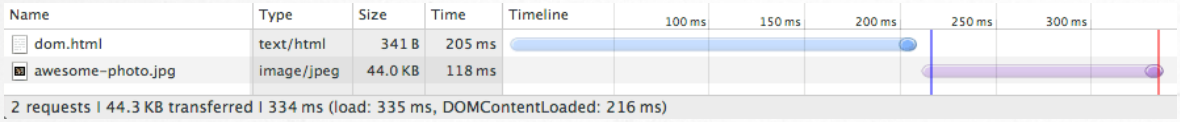
- JavaScript может изменять DOM и CSSOM, а также получать данные из этих моделей.
- Выполнение JavaScript невозможно без модели CSSOM.
- JavaScript препятствует созданию DOM, если не использовать асинхронизацию.
- **при выполнении встроенного скрипта создание DOM прекращается, что замедляет визуализацию страницы.**

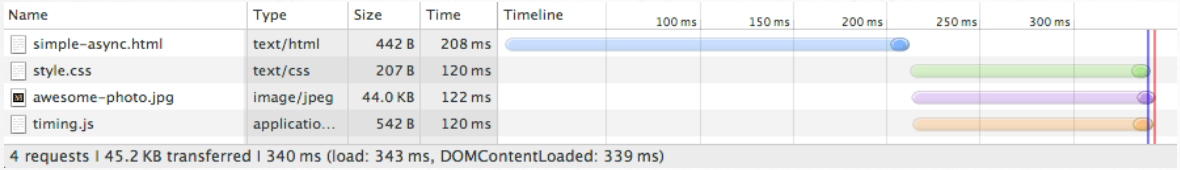
Оптимизация JavaScript для быстрой визуализации страницы

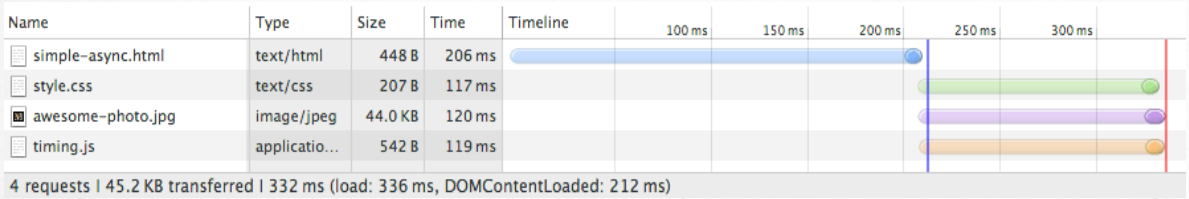
- **Расположение скрипта в документе имеет значение.**
- При обнаружении тега скрипта создание DOM останавливается до тех пор, пока скрипт не будет выполнен.
- JavaScript может получать данные из DOM и CSSOM, а также изменять их.
- Выполнение скрипта откладывается до тех пор, пока не загрузится CSSOM.



- **domLoading.** Начальная отметка. В этот момент браузер начинает анализировать первые байты HTML- документа.
- **domInteractive.** В это время браузер завершил анализ документа и создал модель DOM.
- **domContentLoaded.** К этому моменту модели DOM и CSSOM уже готовы, поэтому ничто не мешает запуску скриптов JavaScript. Это значит, что браузер может приступать к созданию модели визуализации.
  - Если ваш фреймворк JavaScript запускается на этом этапе, вы можете отследить, как долго он выполняется, с помощью меток EventStart иEventEnd.
- **domComplete.** К этому моменту обработка завершена, а контент страницы (например, изображения) полностью скачан. Индикатор загрузки перестает вращаться.
- **loadEvent.** Сигнал полного завершения загрузки. Его можно использовать в качестве триггера функций и скриптов, добавив в HTML-код событие onload.







Теперь браузер инициирует событие DOMContentLoaded почти сразу после завершения анализа HTML. Поскольку JavaScript не блокирует анализатор, CSSOM создается одновременно с обработкой скрипта.

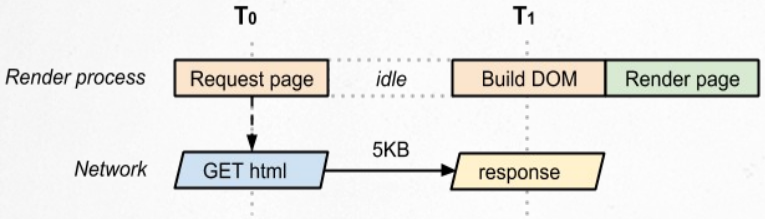
Алгоритмы визуализации

```
<!DOCTYPE html>
<html>
<head>
    <title>CSSOM</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>

<h1>My First Heading</h1>

<p>
    Чтобы визуализировать эту страницу, браузер должен отправить запрос на сервер, скачать HTML-документ,
проанализировать его, создать модель DOM и, наконец, вывести страницу на экран
      <span>My first span</span>
</p>

<img src="http://iteducate.com.ua/wp-content/themes/new-it/relize/img/logo.png" alt="IT Educate">

</body>
</html>
```

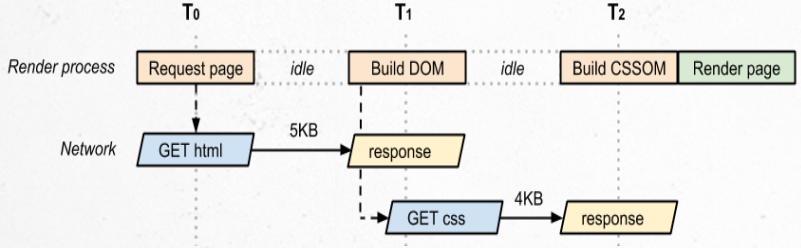
Алгоритмы визуализации

Первоочередные ресурсы - файлы, которые могут заблокировать процесс визуализации.
Продолжительность обработки - количество соединений или общее время, необходимое для загрузки всех первоочередных ресурсов.
Число байтов - количество байтов, которое нужно получить для первоочередной визуализации страницы (общий размер всех первоочередных ресурсов). Вернемся к первому алгоритму.

Алгоритмы визуализации

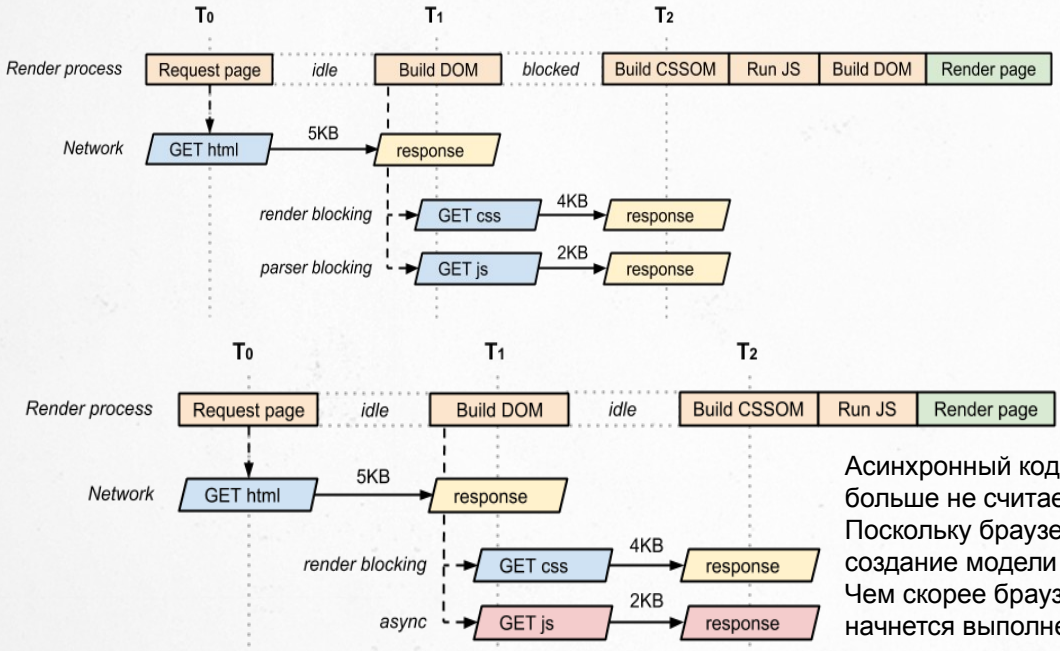


Промежуток между T0 и T1 - это время сетевой и серверной обработки. Если HTML-документ небольшой, то для его загрузки достаточно одного соединения. Большие файлы требуют нескольких соединений из-за особенностей протокола TCP.



первоочередные ресурсы - 2 файла;
продолжительность обработки - 2 соединения минимум;
число байтов - 9 КБ.

Алгоритмы визуализации



Однако на вкладке Network в можно увидеть следующее: когда браузер обнаруживает файлы CSS и JavaScript, он инициирует оба запроса почти одновременно:

- первоочередные ресурсы - 3 файла;
- продолжительность обработки - 2 соединения минимум;
- число байтов - 11 КБ.

Асинхронный код предотвращает приостановку визуализации, поэтому скрипт больше не считается первоочередным ресурсом.
Поскольку браузеру не нужно сразу выполнять скрипт,он может отложить создание модели CSSOM и не блокировать событие DOMContentLoaded.
Чем скорее браузер инициирует событие DOMContentLoaded, тем раньше начнется выполнение скрипта. Новые значения 2 – 2 – 9.

Оптимизация контента

- Удаление ненужных ресурсов
- Оптимизация кодировки и размера передаваемых текстовых ресурсов
- Оптимизация изображений
- Оптимизация шрифтов
- HTTP-кеширование

Минификация: предварительная обработка и оптимизация на основе контекста

- Лучший способ сжать избыточные или ненужные данные - это удалить их. Конечно, мы не можем просто так стирать информацию, но в некоторых случаях, зная о формате данных и его свойствах, всегда можно значительно снизить размер ресурса, не меняя его суть.

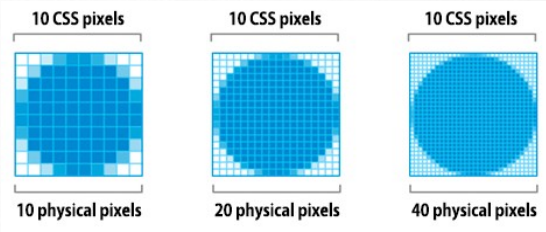
Сжатие текста с помощью GZIP

- GZIP - это стандартный компрессор, который может быть применен к любому потоку байтов. Он запоминает встреченный ранее контент, а затем находит и заменяет повторяющиеся фрагменты данных. GZIP лучше всего сжимает текстовые ресурсы, часто достигая коэффициента сжатия 70-90% при работе с большими файлами.
- GZIP лучше всего сжимает текстовые ресурсы: CSS, JavaScript и HTML.
- Все современные браузеры поддерживают и автоматически запрашивают сжатие GZIP.
- На сервере должно быть настроено сжатие GZIP.
- В некоторых сетей доставки контента вы должны проверить, включено ли сжатие GZIP.

Библиотека	Размер	Размер после сжатия	Коэффициент сжатия
jquery-1.11.0.js	276 КБ	82 КБ	70%
jquery-1.11.0.min.js	94 КБ	33 КБ	65%
angular-1.2.15.js	729 КБ	182 КБ	75%
angular-1.2.15.min.js	101 КБ	37 КБ	63%
bootstrap-3.1.1.css	118 КБ	18 КБ	85%
bootstrap-3.1.1.min.css	98 КБ	17 КБ	83%
foundation-5.css	186 КБ	22 КБ	88%
foundation-5.min.css	146 КВ	18 КБ	88%

Удаление и замена изображений

- Удалите ненужные изображения.
- При возможности применяйте эффекты CSS3.
- Используйте веб-шрифты вместо кодировки текста в изображениях.



Оптимизация для экранов с высоким разренением

- На экранах с высоким разрешением один CSS-пиксель состоит из нескольких экранных пикселей.
- В изображениях высокого разрешения пикселей и байтов гораздо больше, чем в обычных.
- Техники оптимизации можно применять к изображениям любого разрешения.

Оптимизация шрифтов

- Шрифт состоит из глифов - векторных форм каждой буквы или символа. Поэтому размер файла шрифта зависит от двух переменных: количества глифов в шрифте и сложности их векторных контуров. Например, Open Sans, один из самых популярных веб-шрифтов, содержит 897 глифов, включая латинские, греческие и кириллические символы. Сегодня в Интернете используются четыре формата контейнеров шрифтов: **EOT, TTF, WOFF и WOFF2**. К сожалению, несмотря на возможность выбора, не существует единого формата, который работает во всех браузерах. Например, EOT доступен только в IE, TTF поддерживается в этом браузере только частично. WOFF распространен шире всего, однако его нельзя использовать в некоторых старых браузерах, а над поддержкой WOFF 2.0 работают в настоящее время.

Создание семейства шрифтов с помощью @font-face

- Используйте подсказку format(), чтобы указать форматы шрифтов.
- Чтобы ускорить работу сайта, разделяйте крупные Unicode-шрифты на поднаборы. Используйте субнастройки Unicode-диапазона и вручную добавьте запасной вариант для более старых браузеров.
- Чтобы ускорить работу сайта и отрисовку страниц, сократите количество вариантов стиля для каждого шрифта.

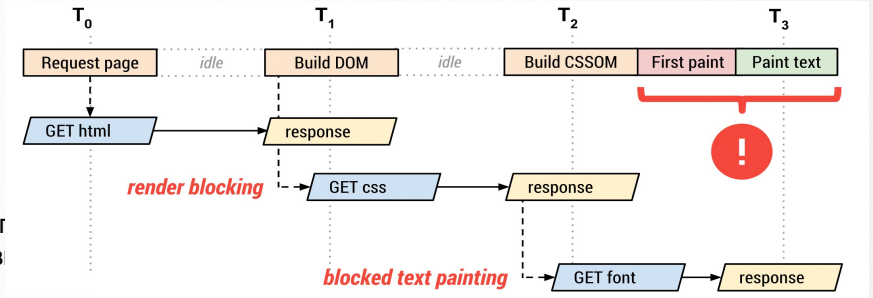
Оптимизация шрифтов

- Браузер читает разметку страницы и определяет, какие варианты шрифтов нужны для отрисовки текста.
- Браузер проверяет, не установлены ли нужные шрифты на устройстве.
- Если файла нет на устройстве, браузер читает список внешних расположений:
- Если формат указан, перед скачивание браузер проверяется, поддерживается ли он. В случае отрицательного ответа программа переходит к следующему варианту.
- Если указание на формат отсутствует, браузер скачивает ресурс.

Загрузка шрифта

- Запросы шрифтов отправляются только после создания модели визуализации, поэтому отображение текста может быть задержано.
- Font Loading API позволяет изменить исходные настройки отложенной загрузки и использовать собственные стратегии загрузки и отображения шрифтов.
- С помощью встраивания шрифтом можно изменить исходные настройки отложенной загрузки в более старых браузерах.

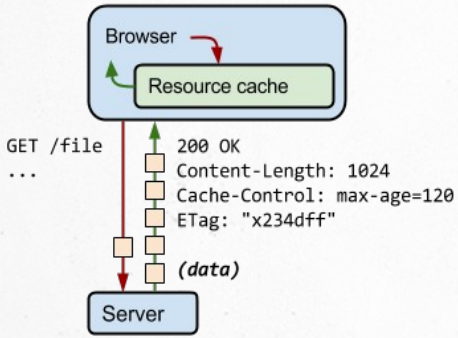
Оптимизация шрифтов



- Браузер запрашивает HTML-документ.
- Браузер анализирует HTML-ответ и создает модель DOM.
- Браузер обнаруживает ресурсы CSS, JavaScript и т. д. и отправляет зап
- После получения CSS-контента браузер создает модель CSSOM и сов визуализации.
  - После того как модель визуализации определила необходимые варианты шрифтов, отправляются соответствующие запросы.
- Браузер отображает макет страницы и ресурсы на нем.
  - Если шрифт ещё не доступен, браузер может не показывать текст.
  - Как только шрифт становится доступен, браузер отображает текст.
  - Между началом показа контента на странице, которое происходит вскоре после создания модели визуализации, и запросом к шрифту идет “гонка”.

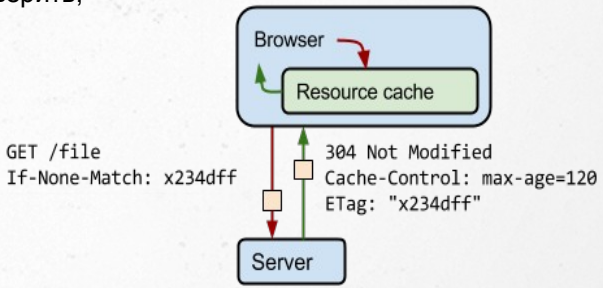
Кеширование

- Скачивать ресурсы страницы заново при каждом посещении - это очень неудобно. Из-за повторных отправок запроса сайт может работать медленно. Кроме того, пользователю придется зря тратить большое количество трафика. Именно поэтому кеширование данных имеет огромное значение при оптимизации сайта. Во всех браузерах есть встроенный HTTP-кеш. Нам осталось убедиться, что в каждом ответе сервер указывает правильные директивы HTTP-заголовков. Они нужны, чтобы указать браузеру, когда и на какой период нужно кешировать ответ.



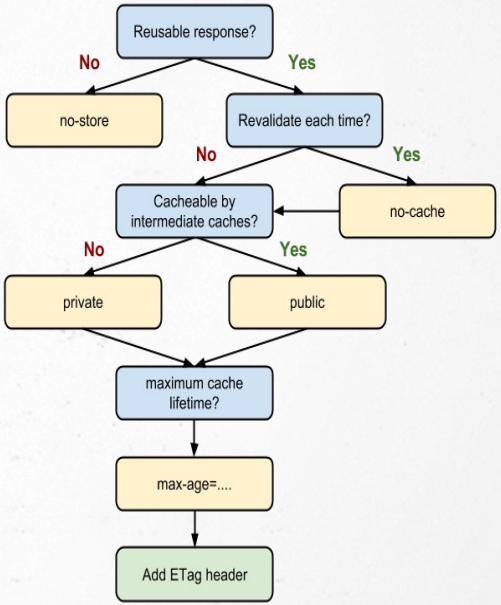
Когда сервер возвращает запрос, он также отправляет набор HTTP-заголовков, описывающих тип контента, длину, команды для работы с кешем, маркер подтверждения и т. д. Например, в примере выше сервер возвращает запрос размером 1024 Б, отдает команду клиенту кешировать его на 120 секунд и отправляет маркер подтверждения (x234dff). Он используется, чтобы проверить, не изменился ли ресурс, после того как срок действия ответа истек.

- Сервер отправляет маркер подтверждения в HTTP-заголовке ETag.
- С помощью маркера подтверждения можно проверить,
- изменился ли ресурс.



Кеширование

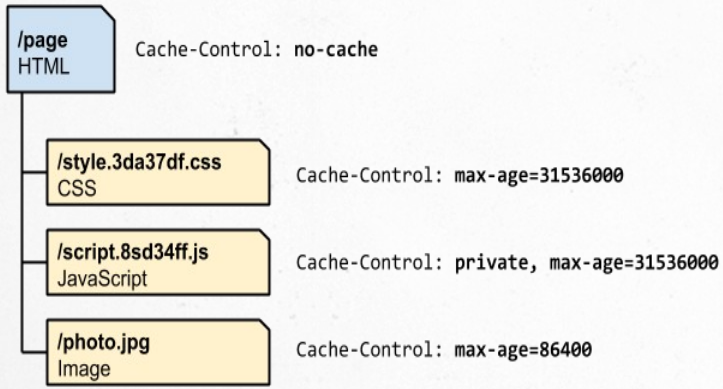
- Лучше всего, если запрос вообще не приходится отправлять на сервер. Используя местную копию ответа, можно избежать задержки при работе сайта и не скачивать лишние данные. Для этого спецификация HTTP позволяет серверу вернуть несколько директив Cache-Control, определяющих, как и на какое время ответ может быть сохранен в кеше браузера и других промежуточных кешах.
- Правила кеширования ресурса можно указать с помощью HTTP-заголовка Cache-Control.
- Директивы Cache-Control определяют, где, как и на какое время может быть кеширован ресурс.



Директивы Cache-Control	Значение
max-age=86400	Ответ можно сохранить в браузере и промежуточных кешах (для него указана директива"public") на 1 день (60 секунд x 60 минут x 24 часа)
private, max-age=600	Браузер может кешировать ответ на 10 минут (60 секунд x 10 минут)
no-store	Ответ нельзя кешировать. При повторном запросе нужно скачать его заново.

Аннулирование и обновление кешированных ответов

- Ответы, сохраненные в локальном кеше, можно использовать, пока не истечет срок действия ресурса.
- Чтобы обновить версию ответа, добавьте в URL файла соответствующую идентификационную метку.
- Чтобы производительность приложения оставалась высокой, определите для него иерархию кешей.



Для HTML указана директива no-cache, поэтому браузер будет проверять актуальность документа при каждом запросе и при изменениях скачивать последнюю версию ресурса. Кроме того, мы изменили HTML-разметку, добавив идентификационные метки в URL CSS- и JavaScript-ресурсов. Если эти файлы изменятся, HTML тоже обновится, и браузер скачает новую копию HTML-ответа. Браузерам и промежуточным кешам разрешено сохранять CSS. Срок его действия заканчивается через 1 год. Обратите внимание, что мы можем установить такой длинный период, потому что мы добавили идентификационную метку в название файла. Поэтому если CSS обновится, то URL тоже изменится. Срок действия для JavaScript тоже истекает через 1 год. Однако ресурс отмечен директивой private, потому что в нем содержатся личные данные пользователи, которые нельзя сохранять в кеше сетей доставки контента.
У кешированного изображения нет версии или уникальной идентификационной метки, и срок его действия заканчивается через 1 день.