Оптимизация WEB приложений

Лекция 3

Оптимизация изображений

За счет использования подходящих графических форматов и эффективного сжатия без потерь суммарный размер страницы может быть уменьшен иногда на 50% и более.

Изображения, полученные с фотоаппаратов или сохраненные в некоторых графических редакторах, могут содержать много килобайт дополнительной информации, комментариев (т. н. метаданных), а также избыточную цветовую палитру.

11.10.2023

Существует несколько графических форматов, поддерживаемых всеми современными браузерами: PNG-8, PNG-24, JPEG, GIF, ICO. Каждый из этих форматов позволяет в определенных ситуациях получить значительный выигрыш в размере по сравнению с другими форматами. Основные рекомендации для различных типов изображений:

- Полноцветные изображения, изображения с градиентами

Для полноцветных изображений с богатой цветовой палитрой (фотографий, сложных градиентов и т. п.) следует использовать формат JPEG высокой степени качества. Необходимо помнить о том, что JPEG — формат сжатия с потерями, и чем выше степень сжатия, тем большее число артефактов появится на итоговом изображении.

- Полупрозрачные изображения

В случаях, когда для верстки требуются полупрозрачные изображения, следует использовать формат PNG-24, поддерживающий альфа-каналы.

Нельзя, однако забывать о том, что браузер Internet Explorer 6 не поддерживает полупрозрачность в таких изображениях и для их корректного вывода следует применять фильтр AlphaImageLoader.

- Изображения с ограниченной цветовой палитрой

Для изображений с ограниченной палитрой следует применять формат PNG-8. Этот формат, как и формат GIF, позволяет использовать прозрачность (не альфа-каналы), но в большинстве случаев превосходит GIF по качеству сжатия итогового файла. Достигается это за счет более совершенной методики сжатия (фильтрации), которая охватывает и горизонтальные, и вертикальные повторения, а также хорошо работает с градиентами.

- Анимированные изображения

Единственным кроссбраузерным форматом, позволяющим отображать анимацию в изображениях, является формат GIF. Однако уже в ближайшем будущем ему может составить конкуренцию развивающийся формат APNG.

- Иконка веб-сайта (favicon.ico)

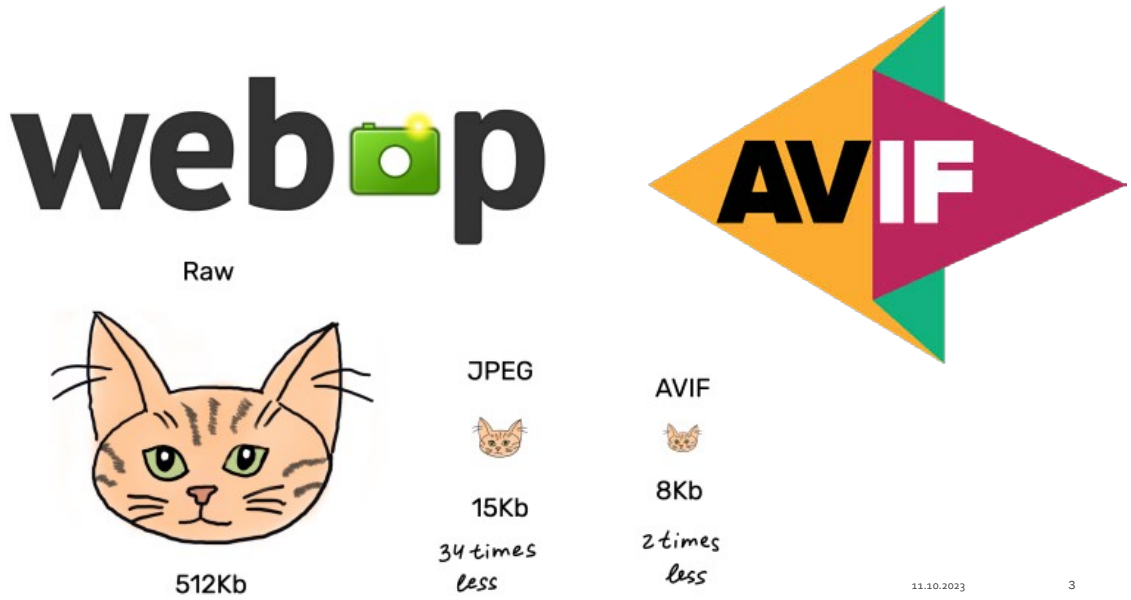
Для иконок веб-сайта существует специальный формат ICO, однако большинство современных браузеров могут использовать в качестве иконки изображение любого поддерживаемого ими формата.

Наибольшего эффекта от оптимизации можно достичь, только используя наиболее подходящий формат для каждого случая. Нередко разделение сложных изображений (содержащих, например, полноцветное изображение с некоторым количеством мелкого текста и полупрозрачную рамку) на несколько отдельных изображений, накладывающихся одно на другое, может существенно уменьшить размер веб-страницы.

Удобной программой для ручной оптимизации статических изображений в Windows и Mac OS является программа Adobe Photoshop, для анимированных изображений — Adobe Fireworks. В операционных системах семейства Linux одним из наиболее подходящих приложений является Gimp.

Современные форматы изображений

PNG и JPG всегда были основными форматами изображений для наших сайтов. Однако, появились форматы нового поколения, такие как AVIF или WebP, чтобы конкурировать с ними.



WebP и AVIF – это форматы следующего поколения, которые предназначены для создания файлов меньшего размера при сохранении высокого качества изображения. Google представил нам WebP в 2010 году, тогда как AVIF на много моложе. Он был создан только в 2019 году.

WebP был изначально запущен в качестве замены традиционных форматов изображений JPEG, JPG и PNG. Поскольку он существует намного дольше, WebP поддерживается большинством браузеров и платформ:

WebP отлично подойдет для ярких, сочных и красочных фотографий, так как он может отображать большее количество пикселей. Кроме того, этот формат обеспечивает очень быстрое времена загрузки, в сравнении с PNG и JPEG. Файлы WebP на 26% меньше, чем файлы PNG, и на 34% меньше, чем файлы JPEG.

С другой стороны, AVIF, производный от видеокодека AVI, является отличным выбором для “lossy compression” (сжатие с потерями) — это когда из файла удаляются ненужные данные, обычно без заметного падения качества. Размеры файлов AVIF очень малы, что обеспечит увеличение скорости загрузки вашего контента. AVIF является лучшим выбором для видео, анимаций и изображений с прозрачным фоном.

Сжатие изображений является одним из наиболее важных аспектов при выборе между AVIF и WebP. С маленьким размером файла можно увеличить скорость загрузки и снизить время Large Contentful Paint (LCP). В свою очередь, вы сможете обеспечить лучший пользовательский опыт (UX) и получить более высокий рейтинг в поисковой выдаче.

WebP был представлен как способ достижения лучшего качества изображения с размером файла, похожим на JPG:

Однако, как и с JPG, вы иногда можете столкнуться с неприятными побочными эффектами, такими как блочность, размытость и цветовой оттенок.

С другой стороны, с AVIF вы можете получить более гладкое, четкое изображение с тем же размером файла. Кроме того, редко возникают любые из вышеупомянутых негативных эффектов. Это является преимуществом AVIF для изображений с высокой детализацией. Кроме того, если на вашем изображении есть текст, то AVIF подойдет лучше, поскольку на фотографии нужна детализация, с которой этот формат справляется отлично.

Устранение избыточного кода

Оптимизация верстки

Для уменьшения размера когда следует использовать такие способы верстки, которые требуют минимум тегов HTML и правил CSS.

Так, семантическая верстка с применением независимых блоков более предпочтительна, чем верстка вложенными таблицами с использованием избыточных тегов.

Не стоит использовать в верстке атрибуты HTML и свойства CSS, значения которых подразумеваются по умолчанию, такие, например, как target=”_self”.

Избыточного кода в CSS можно также избежать, приняв стандарт отображения типовых элементов на веб-страницах, таких как заголовки, параграфы, списки, ссылки и т. д. Один раз определив стиль оформления ссылки и параграфа, больше не придется описывать его для каждого нового блока.

Устранение встроенного в разметку кода

Суммарный объем кода можно также сократить за счет устранения встроенного на веб-странице CSS- и JS-кода. Множество одинаковых атрибутов style="" в HTML-тегах за счет использования классов в большинстве случаев можно заменить единственным, общим для всех элементов CSS-селектором, а множество JavaScript-обработчиков (например, обработчиков onclick="", onmouseover="" и др.) — одним-единственным обработчиком. Изменить верстку и JavaScript-логику в подобных ситуациях, как правило, достаточно несложно.

Неиспользуемый код

Нередко на веб-страницах можно найти некоторое количество неиспользуемого кода, находящегося как в самом HTML-документе, так и во внешних файлах.

Время загрузки этих страниц увеличивается на время загрузки неиспользуемых внешних файлов из сети или из кэша браузера и на время, необходимое для разбора всех элементов DOM-дерева и CSS-правил, которые могут быть к ним применены. В случаях, когда размер веб-страницы и файлов ресурсов измеряется в сотнях килобайт, задержка может быть существенной.

Если в файлах CSS и JS, подключаемых на веб-странице, большой объем кода относится исключительно к другим страницам, следует перераспределить такой код по нескольким файлам, подключая их на страницах по необходимости.

Оптимизация cookie

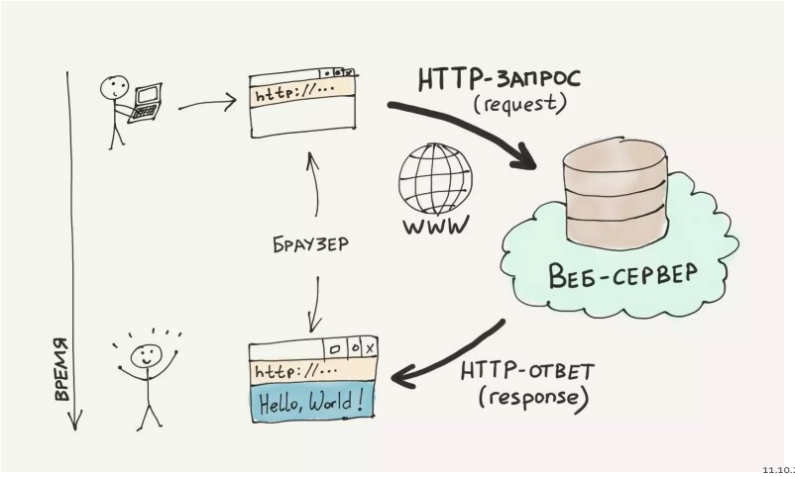
ОПТИМИЗАЦИЯ **COOKIE**

Сохраняя в файлах cookie лишь идентификаторы данных, хранящихся на сервере, можно существенно уменьшить их размер. В идеальной ситуации, чтобы для передачи cookie потребовался лишь один пакет, его размер должен быть не больше одного килобайта.

Сократить размер cookie также можно, гибко определяя содержащиеся в них поля. Если какие-то поля нужны лишь для ограниченного диапазона веб-страниц, нужно определять их только для этого диапазона, а не для всех страниц сайта.

Для статического контента (например, изображений, файлов CSS и JS) можно использовать отдельные домены, не передающие cookie вовсе.

Уменьшение количества запросов



Размер HTML-документа обычно составляет порядка 10% от общего размера страницы. Остальные 90% занимают вызываемые со страницы внешние объекты (чаще всего это изображения, файлы CSS и JS).

Помимо непосредственной загрузки каждого внешнего объекта браузеру необходимо совершить целый ряд дополнительных действий:

- определить IP-адрес сервера по его доменному имени;
- установить новое соединение с этим сервером;
- отработать возможные редиректы;
- отправить запрос на сервер;
- дождаться ответа от сервера.

Таким образом, из-за отсутствия этих временных издержек один внешний объект всегда загружается быстрее, чем несколько объектов того же суммарного размера, загружающихся последовательно, а поскольку многие браузеры загружают внешние файлы JS строго последовательно,

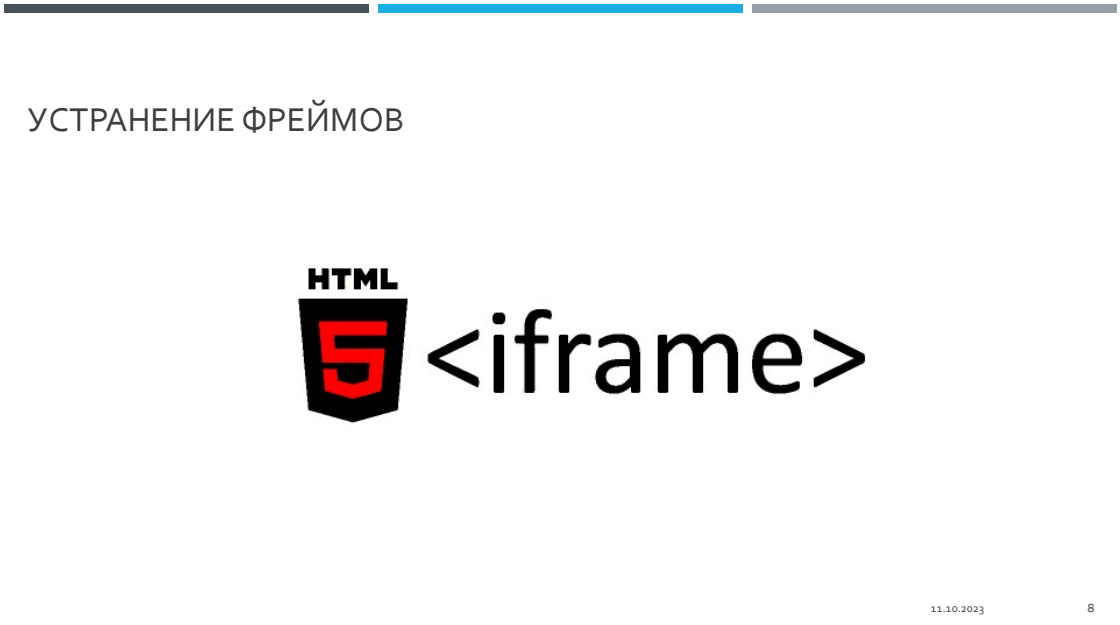
количество этих объектов способно существенно повлиять на скорость загрузки веб-страницы.

Наибольший эффект от уменьшения количества запросов к серверу ощутят пользователи с низкой пропускной способностью канала и большим временем отклика от сервера — обычно это пользователи мобильных устройств и коммутируемых соединений.

Объединение текстовых файлов

Устранение фреймов

На веб-страницах не следует использовать фреймы. При необходимости применения их число должно быть минимальным.



Среди минусов фреймов: избыточные запросы к серверу, блокирование события onload, а также затруднения при поисковой индексации и сохранении адресов и состояний веб-страниц. Следует также заметить, что тег iframe исключён из стандартов XHTML 1.0 Strict и XHTML 1.1.

Использование фреймов, как правило, оправданно только тогда, когда требуется безопасно вставить блок какого-либо стороннего содержимого, например, рекламный блок.

В большинстве же случаев можно избежать применения фреймов за счет разумного использования серверных скриптов и техник AJAX.

Объединение файлов CSS

Уменьшить количество запросов к серверу можно за счет минимизации количества вызовов CSS-файлов. Оптимальное количество таких вызовов — не более двух на страницу.

Не следует подключать на каждой веб-странице все использующиеся на сайте CSS-файлы, пусть тот код, который нужен на ограниченном числе страниц, вызывается только на них.

Разработчики, заботящиеся о доступности своих веб-страниц, часто предусматривают несколько файлов стилей для различных типов устройств просмотра веб-страниц. В этой ситуации в секции <head> может оказаться похожий набор вызовов:

ОБЪЕДИНЕНИЕ ФАЙЛОВ **CSS**

- <link type="text/ css" rel="stylesheet" href="screen.css" media=" screen" />
- <link type="text/ css" rel="stylesheet" href="handheld.css" media="handheld" />
- <link type="text/ css" rel="stylesheet" href="print.css" media="print" />

<link type="text/css" rel="stylesheet" href="screen.css" media=" screen" />

<link type="text/css" rel="stylesheet" href="handheld.css" media="handheld" />

<link type="text/css" rel="stylesheet" href="print.css" media="print" />

ОБЪЕДИНЕНИЕ ФАЙЛОВ **CSS**

```
@media print and (orientation: landscape) {
  .block {
    font-size: 25pt;
  }
}
```


Уменьшить количество запросов в этой ситуации можно, объединив содержимое всех трех файлов в одном и используя для каждого типа устройств отдельное правило @media.

ОБЪЕДИНЕНИЕ ФАЙЛОВ **CSS**

```
@import "fonts.css";
@import "buttons.css";

/* Остальной CSS-код */
```


На этапе разработки, чтобы легче сопровождать код и исключить его избыточность, часто бывает удобно применять CSS-правило @import или вызывать в HTML-документе больше число CSS-файлов. В такой ситуации перед публикацией на сервере можно использовать инструменты для автоматического объединения CSS-файлов и подстановки кода, вызываемого свойством @import, чтобы уменьшить число запросов к серверу.

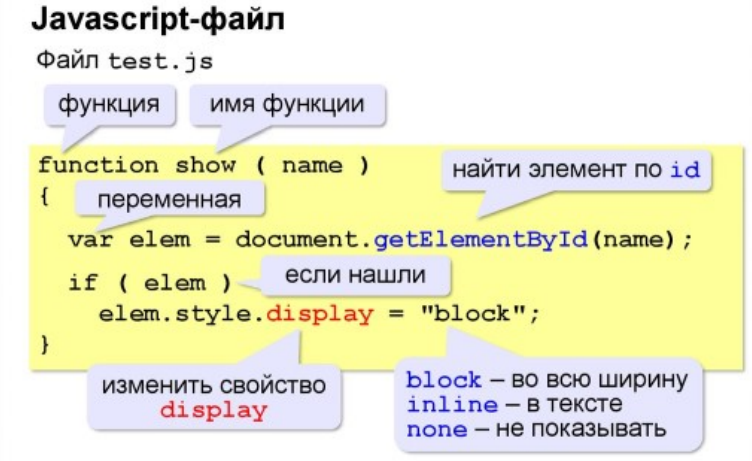
Объединение файлов JavaScript

Как и в рассмотренной выше ситуации с файлами CSS, на странице часто подключается несколько файлов JavaScript. Уменьшив их количество, можно значительно увеличить итоговую скорость загрузки страницы.

ОБЪЕДИНЕНИЕ ФАЙЛОВ **JAVASCRIPT**

Javascript-файл

Файл test.js



Весь код JS можно объединить в одном файле, загружаемом и кэшируемом единожды. Это лучшее решение в том случае, когда кода JavaScript на сайте относительно немного (порядка 50-100 килобайт в сжатом виде).

В ситуации, когда сайт представляет собой сложное веб-приложение и объем кода превышает 100-150 килобайт в сжатом виде, у объединениявсего кода в одном файле имеется отрицательная сторона: при запросе первой страницы пользователь неизбежно будет загружать часть кода, которую мог бы не загружать вовсе. Кроме того, в крупных веб-приложениях бывает не просто уследить за зависимостями модулей, — часто нужный код повторяется и, разумеется, загружается пользователем несколько раз.

Допустим, на сайте существует определенная последовательность страниц, посещаемых каждым новым пользователем, причем для каждой отдельной страницы требуются разные модули JS. Для страницы P1 — модули F1, F2 и F3, для страницы P2 — модули F1, F3 и F4, а для страницы P3 — модули F1, F3, F5 и F6. Возможны три ситуации.

Объекты не объединены в один файл. При загрузке страницы P1 тратится время на загрузку объектов F1, F2 и F3, при загрузке P2 — только объекта F4 (F1 и F3 кэшируются после первой загрузки), а при загрузке P3 — F5 и F6.

Все объекты объединены в один файл. При загрузке страницы P1 тратится время на загрузку объектов F1-F6, но при загрузке всех остальных страниц внешние объекты не запрашиваются.

Объединены только модули, необходимые для текущей страницы.

Для страницы P1 загружаются объекты F1, F2 и F3, для P2 —

F1, F3 и F4, для P3 — F1, F3, F5 и R6. В этом случае каждая отдельно взятая страница при пустом кэше будет загружаться быстрее, однако все три страницы подряд будут загружены медленнее, чем в двух описанных выше случаях, т. к. объекты F1 и F3 дважды повторно загрузятся вместе с другими объединенными объектами.

Выходом из положения может стать продуманное объединение модулей и выделение их ядра, т. е. набора модулей, используемых на большинстве часто загружаемых пользователями страниц. В приведенном примере таким ядром будут объекты F1 и F3.

В сложных ситуациях задача разбиения может быть легко формализована и решена, однако на практике это почти никогда не требуется и для нахождения лучшего варианта объединения достаточно рассмотреть описанные выше варианты.

Более подробно о методах автоматического объединения текстовых файлов рассказано в четвертой главе.

Объединение изображений

Технология объединения изображений для уменьшения числа запросов к серверу достаточно проста. Файл с несколькими объединенными в определенном порядке изображениями (спрайт) загружается единожды, после чего в разных частях страницы отображаются те или иные его области. Возможно даже создание эффектов анимации при помощи объединённых изображений.

Хорошим примером может послужить метод использования единственного изображения для анимированной двухпозиционной кнопки. Верстка такой кнопки представляет собой обыкновенную ссылку с текстом:

```
<a class="button" href="#">Текст кнопки</a>
```

В стилях же, при помощи свойства background и псевдокласса :hover, задано положение фона для различных состояний кнопки:

```
a.button {
background: url(/img/button.png) 0 0 no-repeat; display: inline-block; width: 100px; height: 20px;
}
a.button:hover {
background-position: -100px 0;
}
```

Перед объединением изображений следует разбить их на следующие группы:

- изображения, повторяющиеся по всем направлениям (repeat);
- изображения, повторяющиеся по горизонтали (repeat-x);
- изображения, повторяющиеся по вертикали (repeat-y);
- изображения, не повторяющиеся по вертикали и горизонтали (norepeat).

Эти группы нужны для того, чтобы исключить возможность появления остальных изображений из группы в области другого изображения. Так, изображение с фиксированной высотой, повторяющееся на странице по горизонтали (например, градиентная заливка фона), может находиться в группе аналогичных изображений, выше или ниже их, но никак не слева и не справа.

Если изображение всегда фиксированного размера и не повторяется по какому-либо направлению, его можно размещать в любом месте итогового файла.

Если же изображение, к примеру, использовано в списках, в роли маркера, находящегося в фоне элементов списка, необходимо обеспечить пустое пространство в объединенном изображении правее и ниже изображения маркера. В противном случае под текстом списка может оказаться другая часть спрайта.

Анимированные изображения лучше объединять отдельно, в зависимости от их размера и палитры.

Минусом применения спрайтов является усложнение верстки, однако автоматизация процесса создания спрайтов позволяет устранить этот минус; подробнее об этом написано в четвертой главе. Плюсом же является уменьшение как числа запросов к серверу, так и непосредственно размера страницы в ряде случаев более чем на 40%.

Не стоит забывать и про достаточно старый способ оптимизации, когда несколько расположенных рядом изображений объединяются в одно и при помощи тега <map> на этом изображении размечаются доступные для взаимодействия области.

Встраивание внешних объектов в код веб-страницы Встраивание объектов CSS и JS

В ряде случаев фрагменты CSS- и JS-кода, а также изображения можно подставлять напрямую в HTML-документ.

ВСТРАИВАНИЕ ВНЕШНИХ ОБЪЕКТОВ В КОД ВЕБ-СТРАНИЦЫ ВСТРАИВАНИЕ ОБЪЕКТОВ CSS И JS

- Встраивать код CSS следует внутрь тега <style>, расположенного в секции <head> страницы, а JavaScript-код — в теги <script> в конце документа, перед закрывающим тегом <body> либо также внутри секции <head>.

Плюсы такой подстановки очевидны: количество запросов внешних объектов при загрузке страницы уменьшается до минимального количества. Минусом при использовании такого подхода на всех страницах сайта будет то, что пользователь лишится возможности кэшировать какие-либо объекты, вновь и вновь будет загружать их при просмотре страниц сайта.

Из этого очевидно, что подстановку внешних объектов в HTML-документ следует производить в следующих случаях:

- когда требуется исключительная клиентская производительность страницы, например, на главной странице сайта или других страницах с наивысшей посещаемостью;
- когда суммарная скорость загрузки всех внешних объектов меньше или приблизительно равняется затратам на поочередный запрос каждого из этих объектов по отдельности, т.е. в ситуациях, когда не возникнет уменьшения скорости загрузки страницы из-за невозможности кэширования внешних объектов;
- когда требуется полная автономность веб-приложения.

Встраивать код CSS следует внутрь тега <style>, расположенного в секции <head> страницы, а JavaScript-код — в теги <script> в конце документа, перед закрывающим тегом <body> либо также внутри секции <head>.

Следуя приведенным выше рекомендациям, необходимо избегать встраивания CSS- и JS-кода непосредственно в теги веб-страницы (т. е. в атрибуты style, oncLick и т. д.). Это исключит дублирование кода на странице, а также упростит его сопровождение.

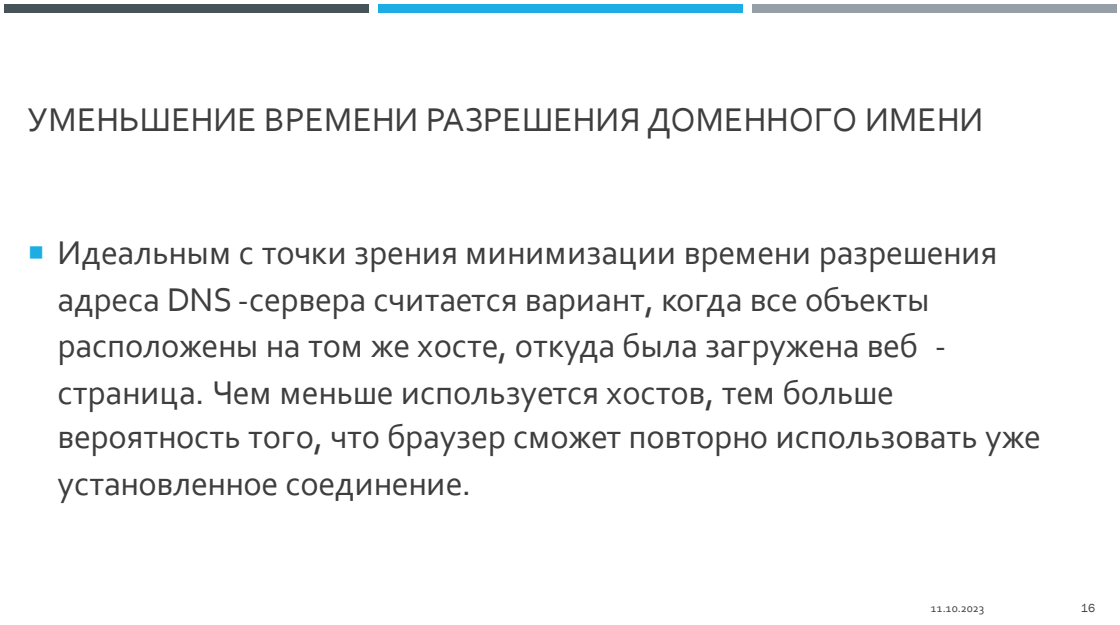
Исключение избыточных HTTP-запросов

Уменьшение времени разрешения доменного имени

Перед тем как браузер сможет установить соединение с веб-сервером, он должен разрешить доменное имя, т. е., зная его, вычислить IP-адрес сервера. Результат может быть закэширован в браузере и операционной системе пользователя, и задержки при повторном открытии веб-страницы не возникнет.

Если же такой записи в кэше не существует, задержка на время

поиска IP-адреса может оказаться значительной и будет зависеть от доступности DNS-сервера, содержащего требуемую информацию (а иногда и от доступности цепочки таких серверов).



Наилучший способ уменьшить временные издержки, связанные с разрешением доменного имени — использовать наименьшее количество различных хостов для размещения внешних объектов веб-страниц, в особенности для тех объектов, которые требуются для первоначального отображения страницы.

Идеальным с точки зрения минимизации времени разрешения адреса DNS-сервера считается вариант, когда все объекты расположены на том же хосте, откуда была загружена веб-страница. Чем меньше используется хостов, тем больше вероятность того, что браузер сможет повторно использовать уже установленное соединение.

На практике же почти у всех браузеров существуют ограничения на количество одновременных соединений с одним хостом. Принимая во внимание это ограничение, наибольший выигрыш в скорости загрузки страниц можно получить, распределив загружаемые объекты по нескольким (4-6) хостам.

Уменьшение количества редиректов

УМЕНЬШЕНИЕ КОЛИЧЕСТВА ЕДИРЕКТОВ

Иногда возникает необходимость перенаправить браузер с одного адреса на другой. Причины чаще всего следующие:

- предоставить пользователю документ, перемещенный на другой адрес;
- позволить пользователю обращаться к документам сайта даже если он ошибся в написании адреса (например, не набрал www в начале адреса);
- направить пользователя на другие домены первого уровня, основываясь на его географическом месторасположении и данных об используемом им языке;
- направить пользователя на определенные страницы в зависимости от того, авторизован он или нет;
- направить пользователя на страницы с другим протоколом (HTTP или HTTPS);
- отследить и сохранить действия пользователя и т. д.

Какой бы ни была причина, каждый редирект порождает дополнительный HTTP-запрос, занимающий определенное время. Поэтому для страниц, для которых скорость загрузки наиболее критична, число редиректов должно быть сведено к минимуму. Для этого необходимо:

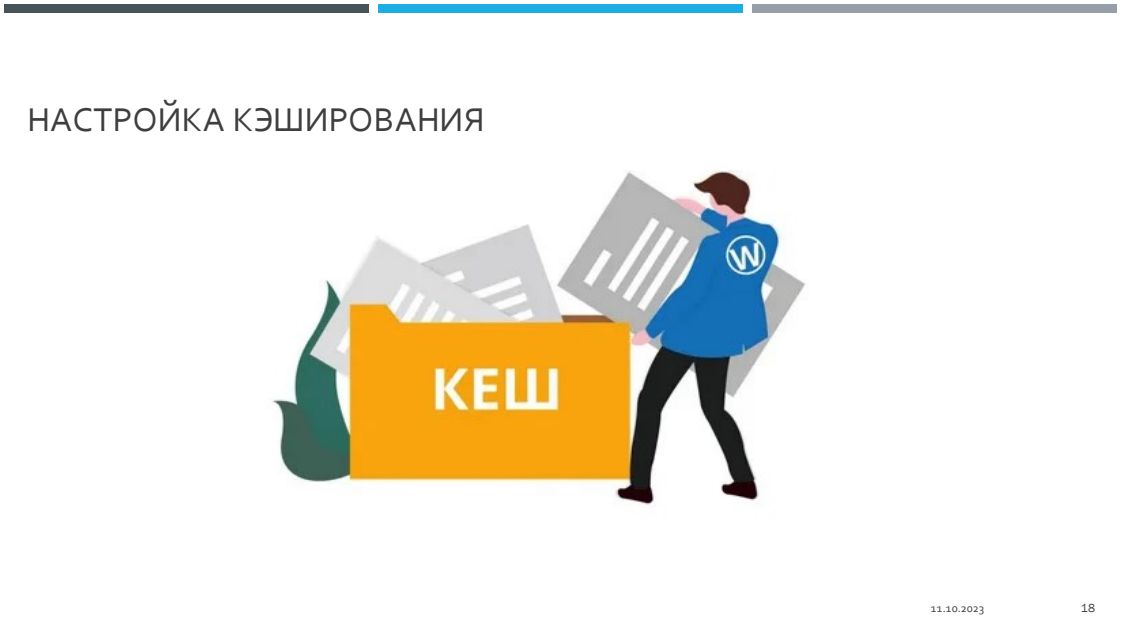
- следить за тем, чтобы ссылки на веб-страницах не вели на адреса, где заведомо будет срабатывать редирект;
- избегать цепных (последовательных) редиректов;
- использовать минимальное количество альтернативных адресов для одних и тех же страниц, стараясь предоставить всем пользователям единственный актуальный адрес для каждой страницы;

- использовать внутренние перенаправления — функцию, доступную в большинстве веб-серверов;
- использовать средства отслеживания информации о пользователе, не основанные на редиректах;
- предпочитать серверные редиректы клиентским, которые могут быть заданы при помощи тега <meta> или JavaScript-обработчика.

Редиректы, отправляющие браузеру код состояния 300, 301 или 302 и заголовок Location, обрабатываются браузером моментально, а при выполнении клиентских редиректов браузеру требуется дополнительное время на разбор полученной веб-страницы.

Кроме того, некоторые браузеры могут кэшировать информацию о редиректах, тем самым ускоряя повторную загрузку ранее открытых веб-страниц.

Настройка кэширования



Браузеры и прокси-серверы обычно стремятся сохранить максимум информации в своих хранилищах, для того чтобы ускорить повторную загрузку ранее загруженных объектов. Важно помнить, что при этом возможна потеря актуальности представляемых данных, поэтому политика кэширования должна быть организована с учетом всех возможных ситуаций.

Кэширование — это один из наиболее мощных механизмов для уменьшения объема передаваемых по сети данных, притом внедряется этот механизм очень просто. Ниже приведено краткое описание наиболее значимых для кэширования заголовков.

Заголовок Expires

Когда HTTP-сервер отправляет объект (например, HTML-документ или изображение) браузеру, он может дополнительно с ответом отправить

заголовок Expires с меткой времени. Браузеры обычно хранят ресурс вместе с информацией об истечении его срока действия в локальном кэше. При последующих запросах к тому же объекту браузер сравнивает текущее время и метку времени у находящегося в кэше ресурса. Если метка времени указывает на дату в будущем, браузер загружает ресурс из кэша, не запрашивая его с сервера. Формат должен быть строго следующим:

ЗАГОЛОВОК **EXPIRES**

Expires: Mon, 27 Dec 2027 00:00:00 GMT

день недели(сокр.), число(2 цифры) месяц(сокр.) год часы:минуты:секунды GMT

Заголовок Expires устанавливает время актуальности информации. Для ресурсов, которые не должны кэшироваться, его нужно выставлять в текущие время и дату (документ устаревает сразу же после получения), для форсирования кэширования его можно определять на достаточно далекую дату в будущем, например:

Expires: Mon, 27 Dec 2027 00:00:00 GMT

Заголовок Cache-Control

Cache-Control: no-store, no-cache, must-revalidate

Cache-Control: max-age=31536000

Заголовок Cache-Control определяет набор директив, относящихся непосредственно ко времени и специфике кэширования документа. Для запрета кэширования можно выставить его в следующее значение:

Cache-Control: no-store, no-cache, must-revalidate

Если же, наоборот, требуется сохранить ресурс в кэш браузера на продолжительный период времени, например, на год (60 * 60 * 24 * 365 секунд), нужно отправлять следующий заголовок:

Cache-Control: max-age=31536000

Заголовки Last-Modified, If-Modified-Since

ЗАГОЛОВКИ LAST-MODIFIED, IF-MODIFIED-SINCE

Last-Modified: Tue, 4 Aug 1995 04:58:08 GMT

If-Modified-Since: Tue, 29 Oct 1994 19:43:31 GMT

Заголовок Last-Modified может отправляться сервером для того, чтобы передать браузеру информацию о дате последнего изменения доку¬мента. Дата должна задаваться в том же формате, что и в случае с заго¬ловком Expires:

Last-Modified: Tue, 4 Aug 1995 04:58:08 GMT

При наличии такой информации в локальном кэше браузер может в следующем запросе отправить ее в заголовке If-Modified-Since:

If-Modified-Since: Tue, 29 Oct 1994 19:43:31 GMT

В случае если дата последнего изменения осталась прежней, сервер ответит кодом состояния 304 Not Modified и данные не будут от¬правлены повторно. В противном случае сервер передаст новую вер¬сию файла.

Данная схема позволяет экономить время, затрачиваемое на переда¬чу данных, однако при ее использовании браузер все равно будет уста-навливать соединение с сервером, чтобы узнать, имеется ли более новая версия.

Заголовки ETag, If-None-Match

ЗАГОЛОВКИ **ETAG, IFNONE-MATCH**

ETag: "any-type-of-tag-or-hash«

If-None-Match: "any-type-of-tag-or-hash"

Заголовок ETag является почти полной аналогией заголовка Last-Modified за тем исключением, что в качестве передаваемого значения может выступать произвольная строка. Заголовок отправляется сервером в следующем формате:

ETag: "any-type-of-tag-or-hash"

Впоследствии, для того чтобы сервер мог определить, является ли объект, находящийся в кэше браузера, точно таким же, как соответст-вующий объект на сервере, браузер может отправить следующий заголовок:

If-None-Match: "any-type-of-tag-or-hash"

И аналогично, если теги совпадают, сервер отвечает кодом состояния 304 Not Modified и данные не передаются повторно. В противном случае сервер передаст новую версию файла.

Форсированный сброс кэша

Если время кэширования при помощи заголовков Expires и Cache-Control установлено на несколько лет вперед и требуется сообщить клиентскому браузеру, что исходный объект изменился, можно воспользоваться двумя способами.

Можно обновить GET-строку запроса, например, используя номер версии или дату последнего изменения:

http://testdomain.com/global.css7v1

http://testdomain.com/global.css720080901

А можно добавить номер версии или дату последнего изменения в само имя файла:

http://testdomain.com/global.v1.css

Во втором случае, для того чтобы исключить проблемы с локальными прокси-серверами, которые могут кэшировать файлы с GET-параметрами, и чтобы не создавать множество физических файлов, достаточно указать в конфигурации сервера правило: при запросах такого вида отдается каждый раз один и тот же файл.

В спецификации RFC-2616 HTTP-кэшированию посвящена целая глава. В ней подробно рассматривается, как работают все приведенные выше заголовки. Также о многих тонкостях использования кэширования более подробно рассказано в четвертой главе.

Увеличение скорости отображения веб-страниц

Даже когда веб-страница и все внешние объекты загружены на компьютер пользователя, браузеру по-прежнему требуется время для того, чтобы разобрать страницу, интерпретировать код HTML и CSS, выполнить код JavaScript. Принимая во внимание особенности работы браузеров на этом этапе, можно достичь существенно более высокой скорости загрузки страницы.

Оптимизация верстки

Разбирая полученный HTML-код, браузер строит дерево документа, содержащее все элементы страницы. Затем, отыскав все взаимосвязи

между элементами этого дерева и CSS-селекторами, относящимися к данной странице, он применяет к документу стили.

Если на веб-странице присутствует большое количество элементов или объем CSS-кода достаточно велик, страница может прорисовываться с ощутимой задержкой. Когда объем кода уменьшить уже невозможно, более высокой скорости загрузки можно достичь за счет эффективной вёрстки. Основные рекомендации к верстке следующие.

- Наиболее важное содержимое страницы должно находиться в самом начале HTML-документа. Так пользователи смогут начать взаимодействовать с этим содержимым раньше.
- Актуальные размеры изображений и ячеек таблиц, содержащих большое количество данных, должны быть явно заданы при помощи HTML-аттрибутов или CSS-свойств. Это позволит избавиться от лишних перерисовок веб-страницы. Например, когда браузер загрузит изображение и определит его размер, ему не потребуется обновлять макет веб-страницы, для изображения уже будет зарезервировано необходимое пространство. Кроме того, точно заданные размеры изображения избавят браузер от избыточной операции масштабирования изображения на лету.
- Следует отказаться от использования CSS-expressions для браузеров Internet Explorer. Expressions отрицательно влияют на производительность браузера и в большинстве ситуаций могут быть заменены более производительным JS-кодом, а иногда и вовсе альтернативной версткой. В ситуациях же, когда для требуемой функциональности сайта использования expressions не избежать, следует применять одноразовые expressions.
- Следует использовать быстродействующие селекторы идентификаторов и селекторы классов. Поскольку большинство браузеров анализируют селекторы справа налево, с виду простой
- селектор #header. menu li a будет применяться дольше, чем аналогичный ему селектор #header. menu-item. В первом случае браузеру необходимо будет найти все ссылки на странице, проверить, находятся ли они в контексте элемента списка, элемента с классом menu и, наконец, элемента с идентификатором header. Второй вариант более предпочтителен, поскольку поиск элементов по классу и идентификатору выполнится существенно быстрее.
- I Универсальные, дочерние, соседние селекторы, селекторы атрибутов, псевдоклассов и псевдоэлементов должны применяться только в тех ситуациях, когда это действительно необходимо. Все эти разновидности CSS-селекторов существенно более ресурсоемки, чем селекторы идентификаторов или классов.

1.5.2. Особенности отображения веб-страниц

При изменении ряда свойств элементов веб-страницы, а также в некоторых других ситуациях браузеры производят перерасчет геометрии и положения элементов, находящихся в потоке веб-страницы, после чего заново перерисовывают страницу или ее часть. Если веб-страница доста-точно

сложна, ее обновление будет заметно для пользователя и неизбежно вызовет задержку в его работе со страницей.

Вот неполный список действий, вызывающих в большинстве браузеров перерисовку страниц или их частей:

- изменение пользователем размера окна браузера или шрифта;
- добавление или удаление CSS-кода (как встроенного, так и во внешнем файле);
- манипуляции с элементами DOM-дерева;
- изменение следующих свойств элементов страницы: class, font, display, visible, margin, padding, width, height;
- активация псевдоклассов, таких, например, как :hover.
- Учитывая эти особенности, можно свести к минимуму число перерисовок страниц, а для того чтобы скорость перерисовок была наивысшей, необходимо:
- обеспечить минимальную глубину и минимальный размер DOM-де¬рева, так как часто изменения свойств какого-либо элемента вынуждают браузер перерисовать не только сам этот элемент, но также родительские и дочерние элементы, а иногда и всю ветвь целиком;
- оптимизировать CSS-селекторы, обеспечить минимальный объем CSS-кода;
- создавать сложные элементы, анимировать их и производить дру¬гие подобные манипуляции над элементами, располагая их вне по¬тока и используя для этого свойства position: absolute и posi¬tion: fixed;
- изменять классы и стили у элементов на максимальной глубине DOM-дерева, что позволит браузеру перерисовывать лишь часть страницы;
- избегать использования таблиц в верстке, поскольку действия над их ячейками почти всегда вызывают необходимость перерасчитать и перерисовать всю таблицу целиком.

Оптимизация структуры веб-страниц

От структуры HTML-документа во многом зависит скорость загрузки страницы пользователем.

Даже при одинаковом суммарном размере страниц и равном количестве внешних объектов две различные страницы могут загружаться за совершенно разное время.

Причина в том, что отображение элементов частично загруженной страницы в большинстве браузеров осуществляется только после выполнения следующих шагов:

1. получения HTML-документа;
2. получения всех внешних объектов CSS, вызываемых в HTML-документе;

3. получения всех внешних объектов JavaScript, вызываемых в HTML-документе внутри тега <head>;

4. получения всех внешних объектов JavaScript в HTML-документе внутри тега <body>, расположенных в потоке выше выводящегося элемента.

1.6.1. Особенности загрузки браузерами внешних объектов Особенности загрузки кода CSS

В большинстве браузеров отображение страницы будет приостанов¬лено до тех пор, пока все внешние CSS-файлы не будут загружены. Кроме того, теги <style>, встречающиеся на странице, будут порождать частич-

ную или полную перерисовку страницы, иногда изменяя уже отобразив¬шийся макет, с которым пользователь мог начать взаимодействовать.

Таким образом, более высокой скорости отображения страницы мож¬но добиться, располагая в самом начале веб-страницы, в разделе <head>, все элементы <link>, содержащие вызовы файлов CSS-стилей, а также все встроенные стили, содержащиеся в тегах <style>.

Особенности загрузки кода JavaScript

Из-за того, что код JavaScript может изменять содержимое и макет веб-страницы, браузеры приостанавливают отображение элементов, следующих за JS-кодом, до тех пор, пока код не будет загружен и исполнен. Большинство браузеров при этом приостанавливают даже загрузку внешних объектов, следующих за таким JS-кодом. Однако если на момент начала загрузки JS-файла загрузка внешних объектов уже была начата, они будут загружены параллельно.

Подробный разбор возможной ситуации

МЕДЛЕННЫЙ ВАРИАНТ

```
<head>
<script type="text/javascript" src="script-1.js"></script>
<link rel="stylesheet" type="text/css" href="style-1.css" />
<script type="text/javascript" src="script-2.js"></script>
<link rel="stylesheet" type="text/css" href="style-2.css" />
</head>
```

В приведенном ниже примере на веб-странице загружаются два внешних файла CSS и JS.

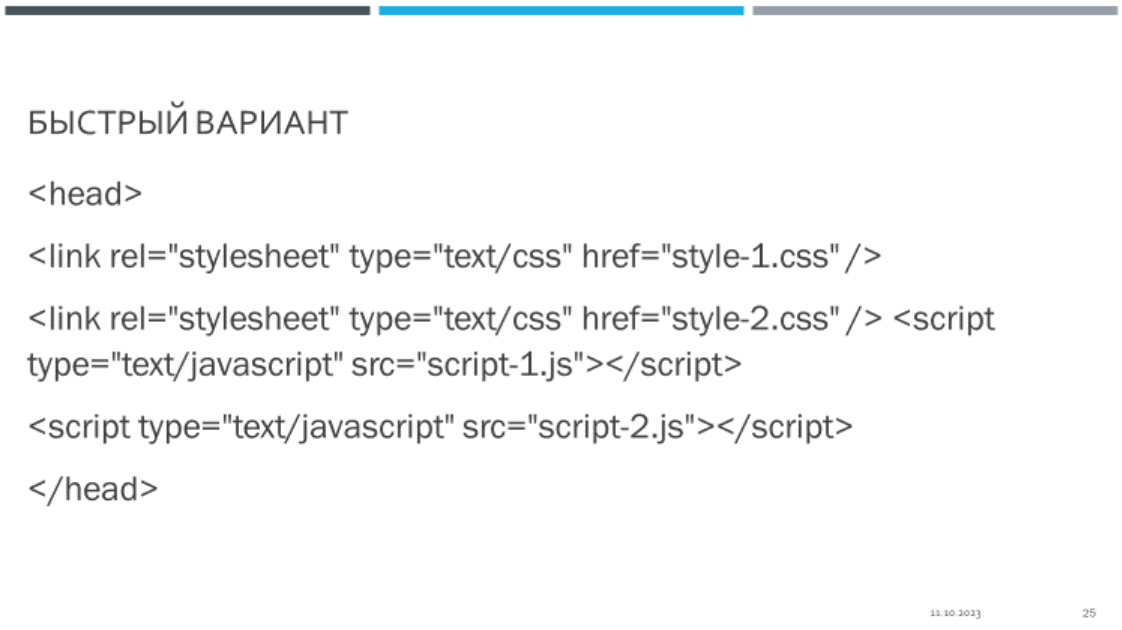
<head>

<script type="text/javascript" src="script-1.js"></script>

<link rel="stylesheet" type="text/css" href="style-1.css" /> <script type="text/javascript" src="script-2.js"></script>

<link rel="stylesheet" type="text/css" href="style-2.css" /> </head>

При таком порядке следования в документе и при пустом кэше большинство браузеров будет вынуждено дождаться загрузки первого файла JavaScript, только потом начать загрузку следующих двух файлов CSS и JS и лишь по завершении загрузки второго файла JS загрузить последний файл стилей.



Всего лишь изменив порядок следования вызовов внешних объектов так, чтобы впереди были вызовы файлов CSS, можно получить ощутимый выигрыш в скорости загрузки.

<head>

<link rel="stylesheet" type="text/css" href="style-1.css" />

<link rel="stylesheet" type="text/css" href="style-2.css" /> <script type="text/javascript" src="script-1.js"></script>

<script type="text/javascript" src="script-2.js"></script>

</head>

В этой ситуации браузер может инициировать параллельную загрузку сразу трех внешних объектов — двух файлов стилей и первого файла JS. По

окончанию их загрузки браузеру остается загрузить лишь один файл JavaScript, и итоговое время загрузки будет ощутимо ниже.

В первой ситуации загрузка каждого файла JavaScript блокирует получение остальных внешних объектов, создавая дополнительные задержки. Во второй же ситуации вместе с загрузкой первого файла JS происходит параллельная загрузка максимально возможного количества внешних объектов.

Разумеется, экономия времени в каждом отдельном случае будет зависеть от размеров загружаемых объектов и их количества, однако пренебрегать этой особенностью не стоит.

Следует помнить, что данная особенность неактуальна в браузерах, поддерживающих параллельную неблокирующую загрузку внешних объектов. Перечень таких браузеров можно получить при помощи инструментов, описанных чуть ранее в этой главе.