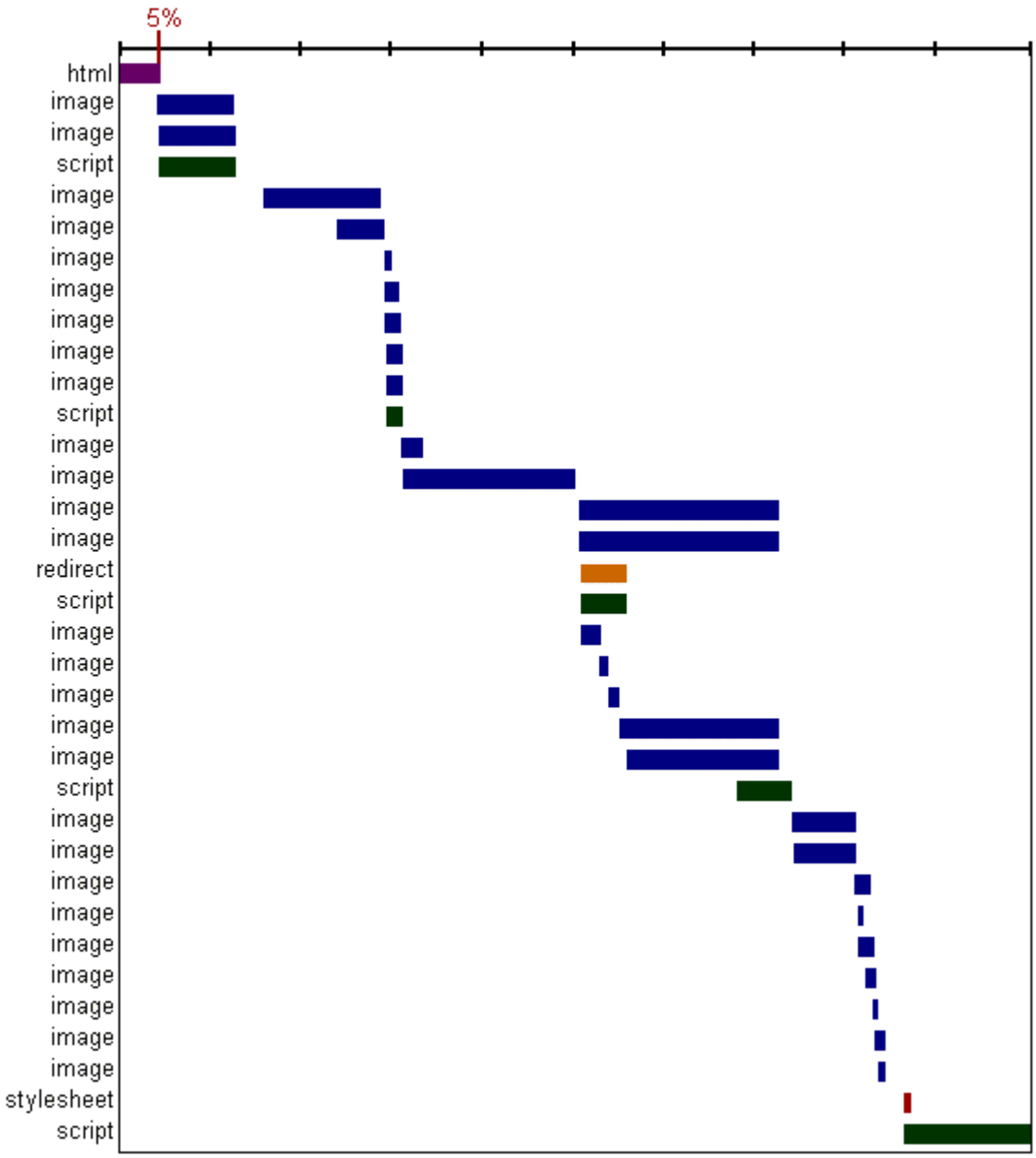
Способы ускорения загрузки Вашего сайта

Автор статьи: Yahoo! Developer Network, Yahoo! Inc. 2008
Оригинал статьи: http://developer.yahoo.com/performance/rules.html
Перевод выполнил: Павел Димитриев, 2008.

СОДЕРЖАНИЕ

Важность производительности фронтенда

В 2004-м году я организовал группу Исключительной Производительности в Yahoo!. Мы являемся небольшой группой разработчиков, чьей целью является улучшение производительности продуктов Yahoo!. Проработав большую часть своей карьеры back-end инженером, я постепенно пришел к этому: я исследовал различные аспекты веб-разработки для выявления наилучших способов увеличения производительности. Т.к. нашей целью является улучшение восприятия продукта пользователем, я измерил время загрузки различных элементов страницы. И вот что я увидел:



На картинке первый ряд, обозначенный «html» - это начальный запрос HTML-документа. В данном случае на его получение было потрачено только 5% всего времени загрузки страницы. Практически для всех сайтов вы увидите то же самое. К примеру, все, кроме одного, сайты, вошедшие в десятку популярных сайтов в Штатах, тратят менее 20% на загрузку HTML. Остальные 80 с лишним процентов времени загрузки тратятся на загрузку того, что указано в самом HTML - собственно, фронтэнда. Вот почему ключем создания быстрых сайтов является улучшение производительности фронтэнда.

Существует три причины, почему стоит начать с производительности фронтэнда:

1. От улучшения фронтэнда больше всего толку: урезав его на половину, мы добъемся 40-процентного (и выше) уменьшения времени загрузки страницы, тогда как уменьшение бекэнда на половину даст в лучшем случае 10% прироста скорости.
2. Оптимизация фронтэнда обычно требует меньше времени и усилий, чем оптимизация бэкэнда (редизайн архитектуры приложения и кода, поиск и оптимизация критичных участков кода, добавления или изменение аппаратного обеспечения, баз данных и пр.)
3. Оптимизация фронтенда зарекомендовала себя: более 50 групп разработчиков в Yahoo! уменьшили время отклика своих приложений на 25% и выше просто следуя нашим советам.

Наше золотое правило: *в первую очередь оптимизируйте фронтэнд - вот где тратится от 80% времени загрузки страницы*.

1: Уменьшите количество HTTP-запросов

80% времени загрузки тратится фронтэндом. Большая часть времени уходит на загрузку компонентов страницы: картинок, таблиц стилей, скриптов, flash.. Уменьшение количества этих компонентов уменьшает количество запросов к серверу, необходимых до того, как клиентское приложение может отрендерить страницу. Это - ключ к созданию быстрых страниц.

Первый путь уменьшить количество компонентов страницы - упростить ее дизайн. А есть ли способ сохранить внешний вид, при этом уменьшив время загрузки? Ниже следует несколько приемов, позволяющих добиться уменьшения количества запросов к серверу, сохраняя функциональность и внешний вид.

Image Map'ы объединяют несколько картинок в одну большую. Общий объем такой картинки примерно равен сумме объемов маленьких картинок, но уменьшение количества запросов к серверу сокращает общее время загрузки страницы. Image Map будет работать, если картинки на странице находятся рядом друг с другом, например в случае полосы навигации. Определение координат для Image Map'ов - занятие довольно утомительное и зачастую приводит к ошибкам (*прим. пер.: вообще-то для этого есть специальный софт, однако ничего не могу сказать про качество его работы*).

CSS-спрайты являются предпочтительным методом сокращения количества запросов на сервер. Объедините все картинки веб-страницы в одну большую картинку и используйте CSS-свойства *background-image* и *background-position* для отображения нужного участка картинки.

Inline-картинки используют URL-схему *data:* для встраивания картинки в саму страницу. Это, однако, увеличит размер HTML-документа. Встраивайте inline-картинки в ваши (кешированные) таблицы стилей - и вы добьетесь уменьшения запросов к серверу, а размер HTML останется прежним.

Объединение файлов также уменьшает количество запросов на сервер. Объедините несколько скриптов в один скрипт, а несколько таблиц стилей - в одну таблицу. Это простая идея, однако она не нашла широкого распространения. Top-10 сайтов по штатам имеют в среднем 7 скриптов и 2 таблицы стилей. Объединение файлов наиболее применимо, когда набор подгружаемых скриптов и CSS отличается от страницы к странице, и этот прием уменьшает общее время загрузки.

Начните оптимизацию своей страницы с уменьшения количества HTTP-запросов. Это самый важный совет для ускорения загрузки страницы у посетителей, которые заходят к вам в первый раз. Как сказано в блоге Tenni Theurer'a: «40-60% посетителей приходят на сайт с пустым кешем». Сделать страницу более быстрой для загрузки

этими пользователями - вот ключ к улучшению восприятия страницы пользователями.

2: Используйте CDN

На скорость загрузки страницы сильно влияет и то, насколько далеко пользователь находится от вашего сервера. Размещение вашего контента между несколькими серверами, разнесенными географически, сделает загрузку сайта быстрее с точки зрения пользователя. Но с чего бы начать?

В качестве первого шага к построению системы с географически распределенным контентом не пытайтесь изменить архитектуру вашего веб-приложения для работы с распределенной архитектурой. В зависимости от приложения, изменение архитектуры может повлечь за собой сложные изменения, такие как синхронизация состояния сессий или репликацию транзакций баз данных между географически разнесенными серверами. Вы можете провалить все попытки сокращения расстояния между серверами и пользователями этими архитектурными изменениями.

Помните, что 80-90% времени загрузки страницы уходит на загрузку ее компонентов: картинок, CSS, скриптов, flash и т.д. Это - *Золотое правило производительности*, как сказано в статье Важность производительности фронтэнда Steve'a Souders'a. Вместо того, чтобы заниматься изменением архитектуры своего приложения, сначала стоит разнести статический контент. Это не только позволяет добиться значительного ускорения загрузки страницы, но также легко реализуется благодаря CDN.

CDN (Content Delivery Network) - это множество веб-серверов, разнесенных географически для достижения максимальной скорости отдачи контента клиенту. Сервер, который непосредственно будет отдавать контент пользователю, выбирается на основании некоторых показателей. Например, выбирается сервер с наименьшим числом промежуточных хопов до него либо с наименьшим временем отклика.

Некоторые крупные интернет-компании владеют своими сетями CDN, однако гораздо дешевле использовать уже готовые решения, такие как Akamai Technologies, Mirror Image Internet либо Limelight Networks. Для стартапов или личных веб-сайтов стоимость услуг сетей CDN может оказаться непомерно высокой, однако по мере того, как ваша аудитория увеличивается и становится все более удаленной от вас, CDN просто необходимы для достижения быстрого времени отклика вашей страницы. В Yahoo! те, кто перенес свой статический контент с серверов веб-приложений в CDN, добились 20-процентного прироста производительности (и выше). Использование CDN потребует лишь незначительных изменений вашего кода, но принесет огромное увеличение производительности ваших веб-приложений.

3: Используйте HTTP-заголовок Expires

Дизайн страниц становится все сложней и сложней, что подразумевает использование большего количества скриптов, CSS, картинок и флеша. Посетителю, пришедшему к вам на страницу в первый раз, возможно потребуется сделать несколько запросов на веб-сервер, однако используя HTTP-заголовок Expires, вы сделаете компоненты страницы кешируемыми. Это предупредит ненужные запросы при последовательном просмотре нескольких страниц. Заголовок Expires чаще всего используется с картинками, однако его следует использовать со *всеми* компоненатами страницы, включая скрипты, флеш, CSS.

Браузеры (и прокси) используют кеш чтобы уменьшить количество запросов к серверу (и их объем) и таким образом загружая страницу быстрее. Веб-сервер использует заголовок Expires, чтобы указать клиентскому приложению, как долго можно хранить объект в кеше. Вот пример заголовка, который указывает, что объект не изменится до 15 апреля 2010 года:

```
Expires: Thu, 15 Apr 2010 20:00:00 GMT
```

Если ваш веб-сервер - Apache, используйте директиву ExpiresDefault, чтобы установить заголовок Expires относительно текущего момента. Следующий пример устанавливает время Expires на 10 лет вперед от момента HTTP-запроса:

```
ExpiresDefault "access plus 10 years"
```

Однако помните, если вы устанавливаете Expires в далекое будущее, вам необходимо изменять название файла компонента страницы каждый раз, когда вы будете менять его содержимое. В Yahoo! это является частью процесса разработки: мы используем версию в названии файла, к примеру yahoo_2.0.6.js.

Установка заголовка Expires в далекое будущее влияет на отображение страницы только для пользователей, которые заходят к вам не впервые. Для тех же, кто зашел на вашу страницу в первый раз, либо кеш его браузера пуст, этот заголовок не сыграет никакой роли. Почувствует ли пользователь сильную разницу в загрузке страницы из-за этого улучшения или нет, зависит от того, насколько часто он заходит к вам с «полным кешем» (кеш уже содержит все компоненты страницы). Мы исследовали это в Yahoo! и оказалось, что 75-80% посетителей уже имеют «полный кеш». Устанавливая заголовок Expires в далекое будущее, вы увеличиваете долю кешируемых компонентов на странице, которые используются повторно (из кеша) при загрузке следующей страницы вашего сайта без пересылки единого байта через интернет.

4: Сжимайте компоненты страницы

Время, необходимое для пересылки через сеть HTTP-запроса и ответа на него, может быть сильно уменьшено решениями разработчиков фронтэнда. Да, конечно, скорость интернета у конечного пользователя, его интернет-провайдер, близость к точкам обмена трафиком - все это вне контроля разработчиков сайта. Но есть также и другие факторы, влияющие на время загрузки страницы. Сжатие помогает уменьшить время загрузки HTTP-ответа, уменьшая его объем.

Начиная с версии протокола HTTP/1.1, веб-клиенты указывают, какие типы сжатия они поддерживают, устанавливая заголовок Accept-Encoding в HTTP-запросе.

```
Accept-Encoding: gzip, deflate
```

Если веб-сервер видит такой заголовок в запросе, он может применить сжатие ответа одним из методов, перечисленных клиентом. При выдаче ответа посредством заголовка Content-Encoding сервер уведомляет клиента о том, каким методом сжимался ответ.

```
Content-Encoding: gzip
```

На данный момент gzip является наиболее популярным и эффективным алгоритмом сжатия. Он был разработан проектом GNU и стандартизован в рамках RFC 1952. Единственный алгоритм, который вы можете встретить где-нибудь, кроме gzip - это deflate, однако он не так эффективен и менее популярен.

В среднем сжатие gzip уменьшает размер HTTP-ответа на 70%. Приблизительно 90% используемых сегодня браузеров указывают, что они поддерживают сжатие gzip. Если вы используете Apache, модуль, отвечающий за сжатие, отличается в разных версиях: Apache 1.3 использует mod_gzip, а Apache 2.x - mod_deflate.

Существуют некоторые несоответствия, когда браузеры (прокси) получают в ответ не то, что ожидают. К счастью, такие случаи встречаются все реже, так как использование старого софта постепенно сходит на нет. Модули Apache выручат вас, автоматически добавляя специальные Vary-заголовки для обхода таких случаев.

Сервер определяет, какие данные нужно сжимать, основываясь на типе файла, но обычно он сильно ограничен в этом выборе. Большинство сайтов сжимают свой HTML. Также стоит сжимать скрипты и CSS, однако многие не используют эту возможность. Фактически, следовало бы сжимать весь контент, который отдается клиенту текстом (в т.ч. XML и JSON). Не стоит жать картинки и PDF, так как они уже сжаты. Попытка сжать их не только отнимет процессорное время, но даже может увеличить размер такого файла.

Сжатие всего, что жмется, - самый простой способ уменьшить страницу в объеме и ускорить ее загрузку к пользователю.

5: Помещайте CSS в начале страницы

Исследуя производительность в Yahoo!, мы пришли к выводу, что помещение CSS в HEAD страницы ускоряет ее загрузку, т.к. позволяет отрендерить ее постепенно.

Разработчики, которые заботятся о производительности своей страницы, всегда хотят, чтобы она могла быть отрендерена постепенно; мы хотим, чтобы браузер мог отобразить любой контент сразу же, как он у него появится. Это особенно важно для страниц, на которых много контента и для пользователей с медленным подключением. Важность визуального оповещения пользователя о текущем состоянии загрузки страницы каким-нибудь индикатором прогресса детально изучена и документирована. В нашем случае в роли индикатора прогресса выступает сама HTML-страница. Когда браузер загружает страницу постепенно - сначала заголовок, потом навигацию, лого наверху и т.д. - все это служит отличным индикатором загрузки для пользователя, который ожидает страницу. Также это улучшает общее впечатление.

Размещение CSS в конце страницы не позволяет начать постепенный рендеринг многим браузерам, в числе которых Internet Explorer. Браузер не начинает рендерить страницу, чтобы не пришлось перерисовывать элементы, у которых во время загрузки изменится стиль. Firefox начинает сразу отрисовывать страницу, в процессе загрузки, возможно, перерисовывая некоторые элементы, но это является причиной появления нестилизованного контента (FOUC - Flash Of Unstyled Content).

Спецификация HTML4 устанавливает, что таблицы стилей должны быть включены в HEAD-секцию документа: «В отличие от A, [LINK] может появляться только в секции HEAD, зато там [LINK] может встречаться сколько угодно раз». Ни одна из альтернатив - белого экрана и показа нестилизованного контента - не стоит риска. Оптимальным решением является следование спецификации и включение CSS в HEAD-секцию документа.

6: Помещайте скрипты в конец

Предыдущее правило рассказало нам, как размещение таблиц CSS в конце страницы тормозит отрисовку страницы браузером и как помещение CSS в HEAD решает эту проблему. Скрипты (внешние .js-файлы) создают похожую проблему, но решается она ровным счетом наоборот: скрипты следует переносить в самый низ страницы, как можно ближе к концу. Делая так, мы позволяем браузеру рендерить страницу постепенно и одновременно распараллеливаем загрузку.

В случае с CSS, постепенный рендеринг не начинается, пока не подгрузятся все таблицы стилей. Вот почему лучше включать их в секцию HEAD, чтобы они грузились в первую очередь и не тормозили рендеринг. В случае скриптов, постепенный рендеринг не начинается для всего контента *ниже* скрипта. Таким образом, нам выгоднее размещать скрипты как можно ниже.

Вторая проблема, порождаемая скриптами - они блокируют параллельную загрузку. Спецификация HTTP/1.1 советует, чтобы браузеры параллельно загружали не более 2-х компонентов веб-страницы с одного хоста. Таким образом, если картинки для вашего сайта располагаются на разных хостах, вы получите более 2-х параллельных загрузок (я заставил Internet Explorer загружать более 100 картинок параллельно). А когда загружается скрипт, браузер не будет начинать никаких других загрузок, даже с других хостов.

Но в некоторых ситуациях совсем не просто перенести скрипты в конец страницы. Например, если скрипт использует *document.write* для вставки части контента на страницу, такой скрипт не получится перенести вниз. Во многих случаях есть способы обхода таких ситуаций.

Альтернативным решением, которое часто используется, является отложенное выполнение скриптов. Атрибут DEFER указывает, что скрипт не содержит document.write и, видя его, браузер продолжает рендеринг страницы. К сожалению, Firefox не поддерживает атрибут DEFER. В Internet Explorer выполнение скрипта можно задержать, но не настолько, насколько хотелось бы. Если выполнение скрипта можно отложить, значит его можно перенести в конец страницы. Это сделает ее загрузку быстрее.

7: Избегайте CSS-выражений (expressions)

CSS-выражения являются мощным (и опасным) способом динамического задания стилей. Они поддерживаются Internet Explorer'ом начиная с Internet Explorer 5. К примеру: вот так можно устанавливать разный фон в зависимости от текущего времени:

```
background-color: expression( (new.Date()).getHours()%2 ?
"#B8D4FF" : "#F08A00" );
```

Как видно из примера, метод *expression* принимает javascript-выражение. Свойство CSS устанавливается равным результату вычисления этого выражения. Метод *expression* игнорируется остальными браузерами, поэтому он полезен для создания кросс-браузерного кода (для создания одного и того же поведения страницы в разных браузерах).

Проблема с этими выражениями в том, что они вычисляются гораздо чаще, чем многие могли бы ожидать. Они вычисляются не только во время рендеринга страницы и изменения размеров окна, но также при скроллинге и даже когда пользователь просто водит мышкой над страницей. Это несложно отследить, достаточно добавить счетчик в выражение. Обычное движение мышкой над страницей запросто может вызвать вычисление выражения более 10000 раз.

Единственный способ избежать огромного числа вычисления CSS-выражений - использование одноразовых выражений, когда после проведения всех необходимых вычислений, они устанавливают свойство CSS-стиля к какому-то конечному статическому значению, заменяя им CSS-выражение. Если вам необходимо динамически изменять свойство CSS-стиля по мере пребывания пользователя на странице, вы можете использовать прием с перехватчиками событий (event handlers) в качестве альтернативы. Если же вам обязательно нужно использовать CSS-выражения на странице, помните, что они могут вычисляться тысячи раз и тем самым повлиять на производительность всей страницы.

8: Выносите javascript и CSS во внешние файлы

Многие из этих правил по улучшению производительности описывают то, как нужно работать с подключаемыми компонентами (в частности, со скриптами и CSS). Однако прежде чем вы начнете применять эти советы на практике, было бы неплохо, если бы вы поинтересовались более базовым вопросом: а стоит ли вообще подключать .js- и .css-файлы или можно включить весь их код непосредственно в код страницы (inline)?

Использование подключаемых файлов на практике обычно дает более быстрые страницы, т.к. браузеры кешируют файлы скриптов и CSS. Код javascript и CSS, который встраивается в HTML, загружается каждый раз, когда загружается сам HTML-документ. Это уменьшает количество необходимых HTTP-запросов, но увеличивает объем HTML. С другой стороны, если скрипты и таблицы стилей находятся в отдельных файлах, скэшированных браузером, размер HTML уменьшается, не увеличивая при этом количество HTTP-запросов.

В таком случае ключевым фактором является частота, с которой кэшируются внешние .js- и .css-файлы относительно количества запросов самого HTML-документа. И хотя этот фактор очень сложно посчитать, его можно приблизительно оценить различными способами. Если ваши пользователи во время одного посещения загружают страницу несколько раз или загружают похожие страницы, которые используют один и тот же код - это лучший случай, чтобы получить все преимущества от вынесения кода в отдельные файлы.

Многие сайты только на половину удовлетворяют этим условиям. Для таких случаев в целом лучшим решением будет создание внешний файлов скриптов и таблиц стилей. Единственное исключение, которое я видел, когда использование inline-кода дает большее преимущество - это использование его на домашних страницах, таких как главная страница Yahoo! (http://www.yahoo.com/) и My Yahoo! (http://my.yahoo.com/). Для страниц, которые загружаются всего несколько (обычно - один) раз за весь сеанс, выгодней встраивать скрипты и таблицы стилей прямо в HTML-документ, чтобы выиграть в скорости загрузки.

Для таких главных страниц, которые открываются первыми в последовательности других с этого же сайта, есть прием снижения количества HTTP-запросов за счет включения скриптов и CSS в код страницы, равно как использования всех преимуществ кэширования за счет динамической загрузки внешних файлов после загрузки всей страницы. Следующие страницы будут использовать уже скэшированные файлы.

9: Уменьшайте количество DNS-запросов

Система DNS устанавливает соответствие имен хостов их IP-адресам, точно так же, как телефонный справочник позволяет узнать номер человека по его имени. Когда вы набираете www.yahoo.com в адресной строке браузера, преобразователь DNS (DNS-resolver), к которому обратился браузер, возвращает IP-адрес узла. DNS имеет свою цену. Обычно требуется 20-120 миллисекунд, чтобы выполнить DNS-запрос и получить ответ (*прим. пер.: в российских реалиях это время обычно выше*). Браузер вынужден ожидать завершения DNS-запроса, т.к. до этого момента он еще не может ничего загружать.

Для повышения быстродействия результаты DNS-запросов кэшируются. Это кэширование может происходить как на специальном сервере интернет-провайдера, так и на компьютере пользователя. Информация DNS сохраняется в системном кэше (в Windows за это отвечает служба «DNS Client Service»). Большинство браузеров имеет свой кэш, не зависящий от системного. Пока браузер хранит DNS-запись в своем кэше, он не обращается к операционной системе для DNS-преорбазования.

Internet Explorer по умолчанию кэширует результаты DNS-запросов на 30 минут, как указано в переменной реестра *DnsCacheTimeout*. Firefox кэширует DNS-ответы на 1 минуту, что видно из установки *network.dnsCacheExpiration* (Fasterfox увеличивает это время до 1 часа).

Когда клиентский кэш очищается (как системный, так и у браузера), количество DNS-запросов возрастает до количества уникальных имен хостов на странице. А это включает в себя собственно адрес самой страницы, картинок, скриптов, CSS, объектов Flash и т.д. Уменьшение количества уникальных имен хостов уменьшает количество DNS-запросов.

Уменьшение количества уникальных имен хостов потенциально уменьшает количество параллельных загрузок компонентов страницы. Уменьшение количества DNS-запросов уменьшает время загрузки страницы, но уменьшение количества параллелльных загрузок может увеличить это время. Мой совет - распределить загружаемые компоненты между 2-4 (но не более) уникальными хостами. Это является компромиссом между уменьшением количества DNS-запросов и сохранением неплохой параллельности при загрузке компонентов страницы.

10: Минимизируйте Javascript

Минимизация скрипта - это удаление из кода всех несущественных символов с целью уменьшения объема файла скрипта и ускорения его загрузки. В минимизированном коде удаляются все комментарии и незначащие пробелы, переносы строк, символы табуляции. В случае с Javascript, это уменьшает время загрузки страницы, т.к. размер файла уменьшается. Две самых популярных утилиты для минимизации javascript - JSMin и YUI Compressor.

Обфускация является альтернативным способом сокращения исходного кода. Также, как минимизация, она удаляет пробельные символы и вырезает комментарии, но в дополнение она измененяет сам код. К примеру, во время обфускации имена функций и переменных заменяются на более короткие, что делает код более компактным, но менее читабельным. Обычно этот прием используется для усложнения реверс-инжиниринга программы. Но обфускация помогает также уменьшить код настолько, насколько это не получится сделать одной минимизацией. С выбором средства для обфускации javascript не все так ясно, но я думаю, что самая распространенная утилита для этого - Dojo Compressor (ShrinkSafe).

Минимизация javascript - безопасный и довольно простой процесс. С другой стороны, обфускация из-за своей сложности может вносить в код баги. Обфускация также требует правки вашего кода для выделения в нем API-функций и других элементов. которые не должны быть изменены. Это также делает более сложной отладку в продакшне. Я никогда не видел, чтобы минимизация кода создавала в нем баги, но вот при обфускации такое случалось. Среди первой десятки американский сайтов при минимизации в среднем достигалось 21% сжатие, тогда как при обфускации - 25%. И хотя обфускация позволяет добиться большего сжатия, я все же рекомендую применять минимизацию кода, так как она не добавит в ваш код ошибок и полученный скрипт проще будет отлаживать.

В дополнение к минимизации внешних скриптов, встроенные в HTML-код скрипты также могут и должны быть минимизированы. Даже если вы сжимаете скрипты gzip'ом, как описано в четвертом правиле, минимизация все равно даст выигрыш от 5% и более. По мере того, как будет увеличиваться ваш javascript-код, будет увеличиваться и процент сжатия от минимизации.

11: Избегайте редиректов

Редиректы осуществляются посредством отправки клиенту статус-кодов HTTP 301 и 302. Вот пример HTTP-заголовка со статус-кодом 301:

```
HTTP/1.1 301 Moved Permanently
Location: http://example.com/newuri
Content-Type: text/html
```

Браузер автоматически перенаправляет пользователя на новый адрес, указанный в поле *Location*. Вся информация, необходимая для редиректа, есть в этих заголовках, тело ответа обычно остается пустым. Результаты редиректов (ни с кодом 301, ни с кодом 302) на практике не кешируется, пока это явно не объявляется заголовком *Expires* либо *Cache-Control*. Другими способами перенаправить пользователя является использование мета-тега Refresh и javascript'а, однако, если вам все же необходимо сделать редирект, предпочтительней использование именно статус-кодов HTTP 301 и 302, хотя бы из-за того, что у пользователя будут правильно работать кнопки «Назад» и «Вперед».

Главое, что нужно помнить при использовании редиректов - это то, что они отнимают время на свое выполнение, а пользователь должен ждать его завершения. Страница даже не может начать рендериться из-за того, что пользователь еще не получил сам HTML-документ и браузер не может начать загрузку остальных компонентов страницы.

Одним из бесполезных редиректов, которые часто используются (и веб-разработчики не стремятся избегать этого) - когда пользователь забывает ввести завершающий слэш (/) в адресной строке в тех случаях, когда он там должен быть. Например, если вы попробуете открыть http://astrology.yahoo.com/astrology, вам вернется ответ с кодом 301, содержащий редирект на http://astrology.yahoo.com/astrology/ (обратите внимание на завершающий слэш). Это исправляется в Apache использованием *Alias* или *mod_rewrite*, или же *DirectorySlash*, если вы используете перехватчики Apache (Apache handlers).

Объединение старого и нового сайтов также часто является причиной использования редиректов. Кое-кто объединяет часть старого и нового сайтов и перенаправляет (или не перенаправляет) пользователя, основываясь на каких-то факторах: браузере, типе аккаунта пользователя и т.д. Использование редиректов для объединения двух сайтов является достаточно простым способом и требует минимального программирования. Но использование редиректов в таких ситуациях усложняет поддержку проекта для разработчиков и ухудшает восприятие страницы пользователями. Альтернативой редиректу является использование модулей *mod_alias* и *mod_rewrite* в случае, если оба URI находятся в пределах одного сервера. Если же причиной появления редиректов является перенаправление пользователя между

разными хостами, как альтернативу можно рассматривать создание DNS-записей типа CNAME (такие записи создают псевдонимы для доменов) в комбинации с *Alias* или *mod_rewrite*.

12: Уберите повторяющиеся скрипты

Включение одного скрипта дважды на одну страницу снижает производительность. Это не так редко встречается, как вы могли бы подумать. Два из десяти наиболее посещаемых сайтов в Америке содержит повторяющийся javascript-код. Два главных фактора, которые могут повлиять на возникновение повторяющихся скриптов - количество скриптов на странице и количество разработчиков. Когда это случается, повторение скриптов замедляет работу сайта ненужными HTTP-запросами и вычислениями.

Повторяющиеся запросы возникают в Internet Explorer'e, но не возникают в Firefox. Internet Explorer дважды загружает один и тот же скрипт, если он включен в страницу два раза и не кэшируется. Но даже если скрипт скэширован, все равно возникает дополнительный HTTP-запрос, когда пользователь перезагружает страницу.

В дополнение к ненужным HTTP-запросам, тратится время на выполнение кода. Повторное исполнение кода происходит в обоих браузерах, не зависимо от того, был ли скэширован скрипт или нет.

Единственным способом избежать повторного включения одного и того же скрипта является разработка системы управления скриптами в виде модуля вашей системы шаблонов. Обычным способом включения скрипта в страницу является использование тега SCRIPT:

```
<script type="text/javascript" src="menu_1.0.17.js"></
script>
```

Альтернативой в PHP можно считать создание функции *insertScript*:

```
<?php insertScript("menu.js") ?>
```

Кроме простого предотвращения включения одного скрипта на страницу дважды, такая функция может выполнять и другие задачи, к примеру, отслеживать зависимости между скриптами, добавлять номер версии в название файла скрипта для поддержки HTTP-заголовков Expires и пр.

13: Настройте ETag'и

ETag'и (Entity Tags - тэги сущностей) - механизм, который используют браузеры и веб-сервера, чтобы определить, является ли объект, находящийся в кэше браузера таким же, как соответствующий объект на сервере (a Entity (сущность) - другое название того, что мы называем компонентами: картинки, скрипты и т.д.). Тэги сущностей были задуманы как механизм для определения актуальности сущности в кэше браузера, что является более гибким подходом, нежели проверка по дате последнего изменения (last-modified). ETag - это строка, которая однозначно идентифицирует конкретную версию компонента. Единственное требование: строка должна быть заключена в двойные кавычки (лапки). Сервер указывает ETag для компонента используя HTTP-заголовок *ETag*:

```
HTTP/1.1 200 OK
Last-Modified: Tue, 12 Dec 2006 03:03:59 GMT
ETag: "10c24bc-4ab-457e1c1f"
Content-Length: 12195
```

Позднее, если браузер хочет определить актуальность компонента, он передает заголовок *If-None-Match* для передачи ETag'а обратно на сервер. Если ETag'и совпадают, ответ от сервера приходит со статус-кодом 304, уменьшая таким образом объем передачи на 12195 байт:

```
GET /i/yahoo.gif HTTP/1.1
Host: us.yimg.com
If-Modified-Since: Tue, 12 Dec 2006 03:03:59 GMT
If-None-Match: "10c24bc-4ab-457e1c1f"
HTTP/1.1 304 Not Modified
```

Проблема ETag'ов в том, что обычно они используют атрибуты, специфичные в пределах одного сервера. ETag'и не совпадут, если браузер загрузит компонент страницы с одного сервера и попробует проверить его с другим сервером - ситуация очень частая, если вы используете кластер для обработки запросов. По умолчанию и Apache, и IIS включают в ETag такие данные, которые вряд ли дадут положительные результат при проверке на актуальность компонента на разных серверах.

Apache 1.3 и 2.x генерирует ETag в формате *inode-size-timestamp*. Даже если один и тот же файл на разных серверах лежит в одной и той же папке, имеет те же права, размер и время, номер его инода будет отличаться от сервера к серверу.

IIS 5.0 и 6.0 имеют похожий формат ETag'ов: *Filetimestamp:ChangeNumber*. *ChangeNumber* - внутренняя переменная IIS для отслеживания изменений в конфигурации самого IIS, и нет

гарантии, что эта переменная будет одинакова на всех серверах, обслуживающих веб-сайт.

В результате ETag'и, которые сгенерирует Apache или IIS для одного и того же файла, будут отличаться на разных серверах. Если ETag'и не будут совпадать, пользователь не будет получать маленького и быстрого ответа с кодом 304 - собственно, зачем ETag'и и разрабатывались; взамен он будет получать стандартный код ответа 200 и далее весь запрошенный компонент. Если ваш сайт находится только на одном сервере, это не будет для вас проблемой. Но если вы используете несколько серверов с Apache или IIS, устанавливающие ETag в соответствии с настройками по-умолчанию, ваши пользователи будут дольше загружать страницы, на серверах будет большая загрузка, нежели могла бы, вы будете тратить больше трафика, а прокси не будут кэшировать ваш контент так, как хотелось бы. Даже если вы установите заголовок *Expires* в далекое будущее, вам все равно будет приходить условный GET-запрос когда пользователь перезагрузит страницу.

Если вы не получаете всех преимуществ, которые предоставляет ETag, тогда лучше совсем отключить его. Тэг *Last-Modified* позволяет проверять актуальность компонента на основании его timestamp'а, а отключение ETag'а позволяет уменьшить заголовки запроса и ответа. Эта статья из базы знаний Microsoft описывает, как отключить ETag в IIS. Если же вы используете Apache, просто добавьте строку

```
FileETag none
```

в конфигурационный файл сервера.

14: Делайте AJAX кэшируемым

Меня часто спрашивают, применимы ли эти советы для Web2.0 приложений? Конечно, да! Это правило - первое, появившееся после внедрения Web2.0 приложений в Yahoo!.

Одно из преимуществ AJAX'а - это то, что он дает моментальный отклик на действие пользователя, т.к. позволяет подгружать данные асинхронно с сервера. Однако при использовании AJAX все еще нет гарантии того, что пользователь не будет бить баклуши в ожидании получения данных с удаленного сервера. Во многих веб-приложениях время ожидания данных пользователем зависит от того, как применяется AJAX. Напрмер, в почтовом веб-клиенте пользователь будет ожидать результата AJAX-запроса по поиску всех писем, удовлетворяющим некоему критерию. Важно помнить, что «асинхронный» еще не значит «мгновенный».

Для улучшения быстродействия, важно оптимизировать результаты AJAX-запроса. Самое главное: вы должны сделать результаты AJAX-запроса кэшируемыми, что обсуждалось в третьем правиле про HTTP-заголовок Expires. Некоторые из правил также работают с AJAX:

* Правило 4: сжимайте компоненты страницы
* Правило 9: уменьшайте количество DNS-запросов
* Правило 10: минимизируйте Javascript
* Правило 11: избегайте редиректов
* Правило 13: настройте ETag'и

Хотя вообще-то третье правило наиболее важно для уменьшения времени отклика. Взглянем на пример. Почтовый веб-клиент может использовать AJAX для загрузки адресной книги пользователя, чтобы обеспечивать автодополнение. Если адресная книга не изменялась с момента последнего визита пользователя, то в этом случае она возьмется из кэша, если заголовок Expires для нее установлен в будущее. Браузер должен быть информирован о том, нужно ли загружать новую версию или можно пользоваться версией из кэша. Этого можно добиться, например, добавляя timestamp времени последней модификации адресной книги в URL AJAX-запроса, хотя бы вот так: *&t=1190241612*. Если адресная книга не менялась со времени последнего визита пользователя на страницу, timestamp будет тот же и браузер использует версию книги из кэша, не делая ненужного HTTP-запроса. Если же пользователь изменял книгу, то timestamp будет другим и браузер сделает запрос на сервер, чтобы взять оттуда новую версию.

И хотя ответы на AJAX-запросы создаются динамически и могут подходить только для одного пользователя, они все равно могут быть

кэшированы. Используя кэширование, ваши AJAX-приложения работают быстрее.