УДК 519.213:257 (519.657)

Е.В. Крылов, В.К. Аникин, А.А. Воронин

# ИССЛЕДОВАНИЕ ПОДХОДОВ К ПОВЫШЕНИЮ БЫСТРОДЕЙСТВИЯ ИНТЕРПРЕТАЦИИ CSS

## Введение

Большой рост количества сайтов в Интернете порождает конкуренцию между сайтами за пользователей. Одним из способов повышения конкурентоспособности, является увеличение скорости работы сайта. Методы повышения скорости работы сайта можно, условно, разделить на серверные и клиентские. Обычно, к серверным методам обращаются тогда, когда сервер, на котором расположен сайт, не выдерживает нагрузку и скорость его работы для каждого пользователя уменьшается. Клиентские методы ускоряют появление информации в браузере пользователя даже при минимальных нагрузках на сервер. Такие методы подходят как новым (малопосещаемым) сайтам там и большим порталам.

Основные методы ускорения работы клиентской части можно разбить на 6 групп (каждая из которых позволяет решить одну из заявленных задач):

- Уменьшение размера объектов.
- Особенности кэширования, которые способно кардинально уменьшить число запросов при повторных посещениях.
- Объединение объектов. Основными технологиями являются слияние текстовых файлов, применение CSS Sprites или data:URI для изображений.
- Параллельная загрузка объектов, влияющая на эффективное время ожидание каждого файла.
- Повышение CSS-производительности, что проявляется в скорости появления первоначальной картинки в браузере пользователя и скорости ее дальнейшего изменения.
- Повышение скорости работы JavaScript.

В данной статье будет рассмотрена CSS производительность. Скорость интерпретации CSS влияет на время появления картинки в браузере пользователя и скорость ее дальнейшего изменения. Целью данной статьи, является исследование быстродействия интерпретации CSS в зависимости от:

- типа CSS привил
- семантики и объема DOM-дерева
- типа DOCTYPE.

## Терминология

**CSS** (англ. *Cascading Style Sheets* — каскадные таблицы стилей) — технология описания внешнего вида документа, написанного языком разметки. Преимущественно используется как средство оформления веб-страниц в формате HTML и XHTML. Стилевые правила, вынесенные в отдельный файл(ы), отвечают за оформление документа относительно устройств отображения.

**DOM** (от англ. *Document Object Model* — “объектная модель документа”) — это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому документов, а также изменять содержимое, структуру и оформление документов.

**DOCTYPE –** определение версии HTML, которая используется на текущей странице

**HTML** (от англ. *Hypertext Markup Language* — “язык разметки гипертекста”) — это стандартный язык разметки документов в Интернете. Все веб-страницы создаются при помощи языка HTML (или XHTML). Язык HTML интерпретируется браузером и отображается в виде документа, удобном для человека.

**XHTML** (англ. *E**x**tensible Hypertext Markup Language* — Расширяемый язык разметки гипертекста) — язык разметки веб-страниц, по возможностям сопоставимый с HTML, созданный на базе XML. Как и HTML, XHTML соответствует спецификации SGML, поскольку XML является её подмножеством. Вариант XHTML 1.1 одобрен в качестве Рекомендации Консорциума Всемирной паутины (W3C) 31 мая 2001 года.

**Веб-обозреватель**, или **браузер** (от англ. *Web browser*) — это программное обеспечение для поиска, просмотра веб-сайтов, то есть для запроса веб-страниц (преимущественно из Интернета), для их обработки, вывода и перехода от одной страницы к другой.

**XML** (англ. *eXtensible Markup Language* — расширяемый язык разметки) — рекомендованный Консорциумом Всемирной паутины язык разметки, фактически представляющий собой свод общих синтаксических правил. XML — текстовый формат, предназначенный для хранения структурированных данных (взамен существующих файлов баз данных), для обмена информацией между программами, а также для создания на его основе более специализированных языков разметки (например, XHTML).

## Выбор типа CSS правил

Если элемент на странице встречается единственный раз, можно с равным успехом использовать id и class для его стилизации. Рассмотрим, как использование id или class влияет на скорость отображения страницы в браузере.

Скорость работы одиночного CSS-правила весьма высока, и даже десятки и сотни их не должны заметно замедлить работу браузеров.

 ISSN 1562-9945

Поэтому, нужно изучать работу нескольких тысяч правил, иначе точность результатов будет малой. Использовать JavaScript для генерации HTML/CSS-кода не представляется разумным, так как тогда придется учитывать еще и скорость работы JavaScript-движка в браузерах, в итоге, эксперимент будет недостаточно чистым. В результате в данном исследовании были сгенерированы статичные файлы (порядка 300 Кб каждый), которые содержали большое число различных CSS-селекторов. Это число подбиралось по нескольким параметрам, в том числе: размер файла и скорость работы HTML/CSS-кода в браузерах (она должна быть достаточно низкой, чтобы файлы в несколько сотен Кб уже заметно тормозили при открытии). Итоговые файлы содержали по 4096 объявлений различных CSS-классов (или различных идентификаторов), HTML-код содержал соответствующее количество блоков, у каждого свой индивидуальный класс (или идентификатор). Дополнительно проверялась скорость работы с простым наследованием узлов (div p, CSS1) и селектор для выбора потомка первого уровня (div>p, CSS2).

## Метод замера

При тестировании браузеров нужно было, во-первых, открыть на клиенте соответствующую данному случаю страницу, а также отследить время на отображение конкретно HTML/CSS-части (понятно, что оно не совпадает со временем открытия всей страницы, которое еще содержит некоторые другие объекты). Для этого была использована простая техника: перед объявлением CSS-блока запоминается текущая метка времени, после окончания HTML-блока, который должен отобразиться, запомненная метка вычитается из текущей. Таким образом, мы получаем время на отработку данных CSS-правил и кода, которыми ими описывается, на клиенте (плюс, возможно, еще какие-то более-менее постоянные расходы, которые нивелируются, если брать относительный, а не абсолютный выигрыш). Конечно, каждую тестовую страницу можно было подгружать в невидимом iframe или даже AJAX-запросом. Но требовалось узнать, фактически, скорость рендеринга браузером CSS-правил и соответствующего кода, а это время будет расходоваться только при отображении страницы в окне браузера. Поэтому подгружаемую страницу нужно отображать на экране (по возможности, максимального размера), чтобы отследить имеющуюся разницу.

## Результаты эксперимента повышения быстродействия за счет выбора типа правил

Ниже приведена таблица 1 с результатами тестов, которые заключаются в среднем времени отображения страницы для различных вариаций селекторов в разных браузерах. Выделено время, меньшее по сравнению с аналогом. Хочется подчеркнуть, что имеет смысл только относительное ускорение использования одних типов селекторов относительно других в пределах одного браузера. Все времена даны в миллисекундах.

Сравнивать абсолютные значения в рамках данного эксперимента не представляется возможным, так как каждому браузеру дополнительно нужно было расположить на странице несколько тысяч “плавающих” блоков с заданными размерами (float:left; width:20px; height:20px, фон для которых и задавался). Эта задача не имеет ничего общего со скоростью работы CSS-селекторов, но может отнимать существенное время у браузера на подготовку изображения страницы на экране (как видно, например, для Opera).

Время в миллисекундах, затраченное браузерами на отображение страницы, для различных комбинаций селекторов

Таблица 1

Табл. 1 – Время в миллисекундах, затраченное браузерами на отображение страницы, для различных комбинаций селекторов

| **Селекторы** | **Firefox 2** | **Opera 9.5** | **Safari 3** | **IE 7** | **IE 6** | **IE 5.5** |
|---|---|---|---|---|---|---|
| p.class | 308 | 5887 | 237 | 82 | 72 | 145 |
| .class | 219 | 6456 | 225 | 78 | 70 | 149 |
| p#id | 349 | 7377 | 338 | 91 | 87 | 156 |
| #id | 214 | 7427 | 220 | 83 | 84 | 159 |
| div>p.class | 519 | 9412 | 247 | 97 | 84 | 158 |
| div>.class | 836 | 12886 | 257 | 95 | 81 | 159 |
| div>p#id | 549 | 10299 | 247 | 105 | 92 | 172 |
| div>#id | 858 | 15172 | 242 | 113 | 91 | 169 |
| div p.class | 827 | 10706 | 256 | 97 | 84 | 161 |
| div .class | 505 | 15864 | 247 | 95 | 86 | 160 |
| div p#id | 772 | 11952 | 247 | 108 | 99 | 177 |
| div #id | 948 | 13306 | 255 | 108 | 95 | 173 |
| div.div p.class | 1001 | 10519 | 263 | 111 | 94 | 165 |
| div.div .class | 1099 | 18742 | 253 | 105 | 92 | 166 |
| div.div p#id | 1161 | 10989 | 266 | 117 | 95 | 181 |
| div.div #id | 1247 | 15816 | 256 | 114 | 100 | 187 |

## Выводы о выборе типа CSS правил

Очевидный вывод, который можно сделать, это преимущество использования #id перед p#id.

Посчитаем коэффициент эффективности для всех браузеров по формуле

$$a_i = \frac{T_{id}}{T_{pid}} \tag{1}$$

где $T_{id}$ – время отображения для селектора #id;

$T_{p_{id}}$ – время отображения для селектора p#id.

На основании данных из таблицы 1 получим:

Firefox 2: $a_1 = 1.63$

 ISSN 1562-9945

Opera 9.5: $a_2$= 0.993
Safari 3: $a_3$= 1.53
IE 7: $a_4$= 1.09
IE 6: $a_5$= 1.03
IE 5.5: $a_6$= 0.98
Рассчитаем среднее по всем браузерам по формуле:

$$\frac{a_1 + a_2 + a_3 + ... + a_n}{n} \quad (2)$$

где $n$ – общее количество используемых браузеров, и получим значение 1.21. Таким образом, средняя эффективность от использования #id вместо p#id составляет 21%.

Также, можно использовать .class вместо p.class и путем аналогичных расчетов, получаем эффективность 7%. Особо стоит обратить внимание на существенное (до 2 раз) ускорение при переходе от CSS1-селекторов к CSS2 (от div p к div>p, в тех браузерах, которые это поддерживают). Следует отметить, что выборка элементов по классу работает, в целом, быстрее, чем по идентификатору (11%). IE всех версий стабильно выполнял все тесты примерно на одном уровне, а при его текущем доминировании ускорение должно выполняться, в первую очередь, для него. Однако, доля IE сокращается, поэтому нельзя не учитывать и другие браузеры.

## Семантика и DOM-дерево

Рассмотрим как быстро браузер создает DOM-дерево в зависимости от наличия в нем элементов с id или class. Для этого мы подготовим 3 набора HTML-файлов. Первый будет содержать 10000 элементов, у которых только часть будет иметь id (количество именованных элементов варьируется от 50 до 10000, необходимо для оценки влияния DOM-дерева). Второй HTML-файл, практически, идентичен первому, только элементы вместо id имеют атрибут class. В третьем наборе в DOM-дереве оставим только элементы с id (т.е. будем изменять само число элементов от 50 до 10000). Все измерения запустим в скрытом iframe, чтобы избежать прорисовки загружаемой страницы на экране.

## Графики влияния DOM-дерева

На рис. 1 приведены графики влияния DOM-дерева на среднюю скорость вывода документа различными браузерами.

На рис. 2 приведены графики для времени выборки одного элемента из дерева (по идентификатору) при наличии в этом же дереве различного числа элементов с идентификаторами. График ID (10000 get) показывает время на 10000 итераций проведения такой выборки. График ID clean (10000 get) отличается тем, что в дереве идентификаторы присвоены не всем элементам, а только заданному числу:

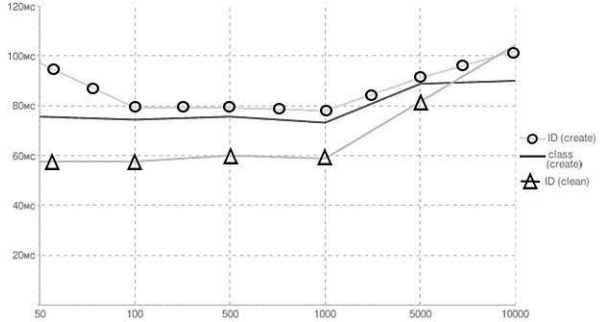


Рис. 1 – Скорость создания документа, среднее по всем браузерам

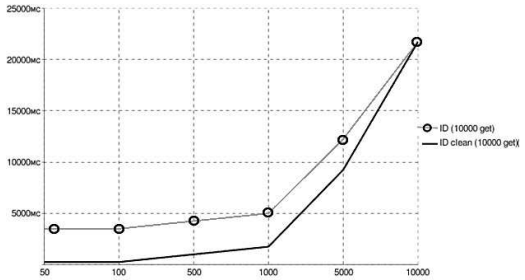


Рис. 2 – Скорость выбора элемента, среднее по всем браузерам

## Выводы по размеру DOM-дерева

По графику средних значений хорошо видно, что, при прочих равных условиях, создание документа с class выходит рациональней, чем с id (в общем случае, от 2% до 10% выигрыша). Если принять во внимание, что .class-селекторы отрабатывают быстрее, чем #id на 7%, то общий выигрыш при использовании в документе классов перед идентификаторами составит порядка 15%. В абсолютном значении эти цифры не так велики: для Centrino Duo 1.7 получается цифра примерно в 0,0085мс на 1 идентификатор (в среднем, 3 CSS-правила и 1 употребление). Для документа со 100 элементами выигрыш может составить почти 1 мс, для документа с 1000 — 8,5 мс! Стоит заметить, что средняя страница в интернете имеет 500–1000 элементов. Проверить, сколько элементов на странице, можно просто запустив следующий код в адресной строке браузера на какой-либо открытой странице: javascript:alert (document.getElementsByTagName(’*’).length) Естественно, что приведенные цифры — это уже то, что даст некоторый ефект. В случае больших веб-приложений задержка в 100 мс (при числе элементов более 10000) уже может оказаться критичной. Ее можно и нужно уменьшать. Что и требовалось доказать, значительную нагрузку составляет именно создание

DOM-дерева в документе. В целом, на эту операцию уходит от 70% всего времени рендеринга (т.е. наибольшая экономия достигается за счет минимизации размера деревьев). На скорость вычисления одного элемента по идентификатору, наибольшее влияние оказывает опять-таки DOM-дерево, чем количество таких элементов. Даже при 1000 элементов с id более половины временных издержек можно урезать, если просто сократить общее число элементов (особенно хорошо это заметно для IE). Можно сформулировать два основных совета: стоит уменьшать DOM-дерево и использовать id только в случае действительной необходимости.

## Семантическое DOM-дерево

Логическим продолжением уже проведенных исследований CSS/DOM-производительности браузеров стало рассмотрение зависимости времени создания документа от числа тегов (узлов дерева). Раздельно были проанализированы случаи, когда DOM-дерево является линейным (все div лежали прямо внутри body), когда оно разветвленное (ветки по 10 вложенных div наращивались внутри body) и когда вместо ветки из div используются некоторая семантическая конструкция, а именно:

```
<div> <ul> <li> </li> <li>  </li> </ul> <p>
<a href="{#}"> <em> </em>
</a> <span>  </span> </p> <blockquote> </blockquote>
<h1>  </h1> </div>
```

Результаты приведены на рис. 3:

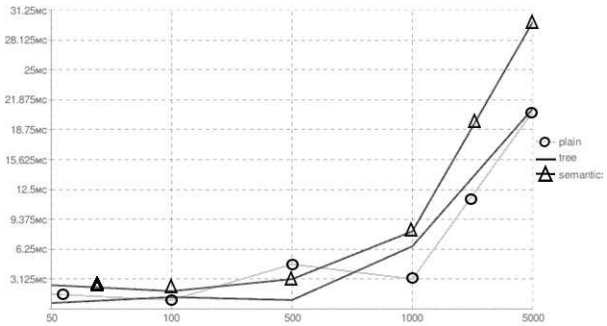


Рис. 3 – Среднее значение времени создания документа от числа узлов в DOM-дереве

## Выводы по исследованию DOM-дерева

Размер DOM-дерева влияет на скорость загрузки страницы. Одной из целей данного исследования было показать, как именно влияет (в конкретных числах). Средний размер страницы — 700–1000 элементов. Они

загрузятся в дерево сравнительно быстро (3–7мс, без учета инициализации самого документа, которая занимает 30–50 мс). Дальше время загрузки растет линейно, но все равно могут возникнуть нежелательные замедления, добавив несколько тысяч скрытых элементов или избыточной семантики. Различия между линейной и древовидной структурой находятся в пределах погрешности, однако, семантическое дерево оказалось самым медленным (чуть ли не на 50%). Но в любом случае, уменьшение размера DOM-дерева всегда является наиболее приоритетным направлением.

## Методика для DOCTYPE

Для проведения эксперимента была использована главная страница Яндекса (она уже хорошо оптимизирована с точки зрения производительности, поэтому проводить эксперименты на ней более показательно). Из нее были удалены все ссылки на картинки и внешние скрипты, чтобы не создавать дополнительных задержек. В дальнейшем полученная “чистая” версия исследовалась различными способами. Далее была добавлена стандартная схема измерения загрузки (рендеринга) страницы: время в самом начале head засекается и затем отнимается от времени срабатывания события window.onload (в данном случае это равносильно окончанию рендеринга HTML-кода). Браузеры друг с другом не сравнивались (в частности, из-за поведения Safari, который не совсем честно сообщает об этом событии), сравнивались только различные варианты. В качестве второй версии страницы бралось приведение ее к валидному XHTML Strict виду. Верстка при этом немного изменилась, но, в целом, результат получился весьма убедительный. Комментарии и пустые атрибуты (пустые onclick=) были сохранены. Размер, несколько увеличился (на 1 Кб не сжатая версия, и на 150 байтов — сжатая). Далее в третьей версии уже были убраны все onclick. Больше ничего со страницей не производилось. Ожиданий данная версия не оправдала (только Safari показал значимые отличия от предыдущего варианта, хотя было удалено 83 пустых onclick). В четвертом варианте — максимуме по чистоте кода (в отношении CSS/HTML-кода) — использование id было сведено к минимуму, все селекторы для class задавались без тегов. Также были убраны все комментарии из кода.

## Результаты эксперимента

В таблице приведены результаты для основных браузеров: указан размер в байтах (Size ) каждого из четырех описанных выше в методике вариантов и время его загрузки в миллисекундах.

## Выводы по DOCTYPE

Целью экспериментов было установить, есть ли различие в отображении HTML 4.0 Transitional и XHTML 1.0 Strict документов. В результате тестов удалось показать, что код соотвествующий стандарту XHTML не медленнее (а даже, местами, быстрее), чем HTML. И оптимизация играет

 ISSN 1562-9945

Таблица 2

Табл. 2 – Размер варианта и время его отображения в миллисекундах

| Size (байт) | IE6 | IE7 | Firefox 2 | Firefox 3 | Opera 9.5 | Safari 3 |
|---|---|---|---|---|---|---|
| 26275 | 56 | 80 | 130 | 127 | 142 | 33 |
| 27173 | 60 | 75 | 127 | 118 | 148 | 27 |
| 26260 | 61 | 75 | 131 | 116 | 141 | 23 |
| 26153 | 55 | 73 | 94 | 102 | 178 | 28 |

роль (возможно ускорение загрузки HTML главной страницы Яндекса на 10–12%). Если говорить о конкретных примерах, то на 100 Кб/с канале с включенным сжатием в FF3 оптимизированный вариант загрузится на 9 мс быстрее. Различие в размере файлов оказалось, минимальным (26153 против 26275 в несжатом варианте, и 8862 против 8845 в сжатом, т.е. меньше 0,5%). При этом в IE7 наблюдается ускорение отображения страницы на 7 мс (от 60–80 мс при загрузке страницы). Это, в среднем, дает 10% выигрыша в скорости. FF3 ведет себя похожим образом (но выигрыш в скорости 20% (25 мс от 127мс)). Все остальные браузеры показали отличие в загрузке на 2–3мс, что укладывается в погрешность.

## Литература

1. Дубаков М. Веб-мастеринг средствами CSS –СПб.:БХВ-Петербург 2002, 518 с.
2. https://developer.mozilla.org/en/Writing_Efficient_CSS
3. http://dean.edwards.name/IE7/usage/optimise.html
4. http://meyerweb.com/eric/css/

*Получено 21.11.2008*