Предисловие

Это подробное руководство по внутренним механизмам работы систем WebKit и Gecko стало результатом обширных исследований, проведенных израильской веб-программисткой Тали Гарсиэль. Она в течение нескольких лет отслеживала всю публикуемую информацию о том, как устроены браузеры (см. раздел Ресурсы), и посвятила много времени анализу их исходного кода. Вот что пишет сама Тали:

Когда на 90% компьютеров был установлен IE, приходилось мириться с тем, что это загадочный "черный ящик", однако теперь, когда более половины пользователей выбирает браузеры с открытым исходным кодом, пришло время разобраться, что скрывается у них внутри, в миллионах строк программного кода на C++...

Тали опубликовала результаты исследования на своем сайте, однако мы считаем, что они заслуживают внимания более широкой аудитории, поэтому размещаем их здесь с некоторыми сокращениями.

Веб-разработчик, знакомый с внутренним механизмом работы браузеров, принимает более квалифицированные решения и понимает, почему следует выбрать те или иные средства. Это достаточно объемный документ, однако мы рекомендуем читать его как можно внимательнее и гарантируем, что вы не пожалеете об этом.*Пол Айриш, Chrome Developer Relations*

Введение

Веб-браузеры, пожалуй, являются самыми распространенными приложениями. В этом учебнике я объясняю, как они работают. Мы подробно рассмотрим, что происходит с момента, когда вы набираете в адресной строке google.ru, до появления страницы Google на экране.

Содержание

1. Введение
    1. Какие браузеры мы рассмотрим
    2. Основные функции браузера
    3. Структура верхнего уровня
2. Модуль отображения
    1. Модули отображения
    2. Основная схема работы
    3. Примеры работы
3. Синтаксический анализ и построение дерева DOM
    1. Синтаксический анализ: общие сведения
        1. Грамматика
        2. Синтаксический и лексический анализаторы
        3. Перевод
        4. Пример синтаксического анализа

5. Формальное определение словаря и синтаксиса
6. Типы синтаксических анализаторов
7. Автоматическое создание синтаксических анализаторов

2. Синтаксический анализатор HTML
    1. Определение грамматики HTML
    2. Контекстная грамматика
    3. DTD в HTML
    4. DOM
    5. Алгоритм синтаксического анализа
    6. Алгоритм лексического анализа
    7. Алгоритм построения дерева
    8. Действия после синтаксического анализа
    9. Обработка ошибок браузерами
3. Синтаксический анализ CSS
    1. Синтаксический анализатор CSS в WebKit
4. Порядок обработки скриптов и таблиц стилей
    1. Скрипты
    2. Ориентировочный синтаксический анализ
    3. Таблицы стилей

4. Построение дерева отображения
    1. Как дерево отображения связано с деревом DOM
    2. Процесс построения дерева
    3. Вычисление стилей
        1. Совместное использование информации о стилях
        2. Дерево правил Firefox
            1. Разделение на структуры
            2. Расчет контекстов стилей с помощью дерева правил
        3. Классификация правил для упрощения сопоставления
        4. Применение правил в порядке приоритета
            1. Порядок приоритета таблиц стилей
            2. Специфичность
            3. Сортировка правил

    4. Последовательное применение
5. Компоновка
    1. Система "грязных битов"
    2. Глобальная и инкрементная компоновка
    3. Синхронная и асинхронная компоновка
    4. Оптимизация
    5. Процесс компоновки
    6. Расчет ширины
    7. Перенос строк
6. Отрисовка
    1. Глобальная и инкрементная отрисовка
    2. Порядок отрисовки
    3. Список отображения Firefox
    4. Хранилище прямоугольников в WebKit
7. Динамические изменения
8. Потоки модуля отображения
    1. Цикл событий
9. Визуальная модель CSS2
    1. Холст
    2. Модель окна в CSS
    3. Схема позиционирования
    4. Типы окон
    5. Позиционирование
        1. Относительное позиционирование
        2. Плавающие элементы
        3. Абсолютное и фиксированное позиционирование
    6. Многослойное представление
10. Ресурсы

Какие браузеры мы рассмотрим

На сегодняшний день существует пять основных браузеров: Internet Explorer, Firefox, Safari, Chrome и Opera. В примерах используются браузеры с открытым исходным кодом: Firefox, Chrome и Safari (код открыт частично). Согласно статистике использования браузеров на сайте StatCounter, на август 2011 года браузеры Firefox, Safari и Chrome были установлены в общей сложности на 60%

устройств. Таким образом, браузеры с открытым исходным кодом имеют на сегодняшний день весьма сильные позиции.

Основные функции браузера

Основное предназначение браузера – отображать веб-ресурсы. Для этого на сервер отправляется запрос, а результат выводится в окне браузера. Под ресурсами в основном подразумеваются HTML-документы, однако это также может быть PDF-файл, картинка или иное содержание. Расположение ресурса определяется с помощью URI (унифицированного идентификатора ресурсов).

То, каким образом браузер обрабатывает и отображает HTML-файлы, определено спецификациями HTML и CSS. Они разрабатываются Консорциумом W3C, который внедряет стандарты для Интернета.
Многие годы браузеры отвечали лишь части спецификаций, и для них создавались отдельные расширения. Для веб-разработчиков это означало серьезные проблемы с совместимостью. Сегодня большинство браузеров в большей или меньшей степени отвечает всем спецификациям.

Пользовательские интерфейсы разных браузеров имеют много общего. Основные элементы интерфейса браузера перечислены ниже.

- Адресная строка для ввода URI
- Кнопки навигации "Назад" и "Вперед"
- Закладки
- Кнопки обновления и остановки загрузки страницы
- Кнопка "Домой" для перехода на главную страницу

Как ни странно, спецификации, которая бы определяла стандарты пользовательского интерфейса браузера, не существует. Современные интерфейсы являются результатом многолетней эволюции, а также того, что разработчики частично копируют друг друга. В спецификации HTML5 не указано, что именно должен содержать интерфейс браузера, однако перечислены некоторые основные элементы. К ним относится адресная строка, строка состояния и панель инструментов. Разумеется, существуют и специфические функции, такие как менеджер загрузок в Firefox.

Структура верхнего уровня

Ниже перечислены основные компоненты браузера (1.1).

1. **Пользовательский интерфейс** – включает адресную строку, кнопки "Назад" и "Вперед", меню закладок и т. д. К нему относятся все элементы, кроме окна, в котором отображается запрашиваемая страница.
2. **Механизм браузера** – управляет взаимодействием интерфейса и модуля отображения.
3. **Модуль отображения** – отвечает за вывод запрошенного содержания на экран. Например, если запрашивается HTML-документ, модуль отображения выполняет синтаксический анализ кода HTML и CSS и выводит результат на экран.
4. **Сетевые компоненты** – предназначены для выполнения сетевых вызовов, таких как HTTP-запросы. Их интерфейс не зависит от типа платформы, для каждого из которых есть собственные реализации.

5. **Исполнительная часть пользовательского интерфейса** – используется для отрисовки основных виджетов, таких как окна и поля со списками. Ее универсальный интерфейс также не зависит от типа платформы. Исполнительная часть всегда применяет методы пользовательского интерфейса конкретной операционной системы.
6. **Интерпретатор JavaScript** – используется для синтаксического анализа и выполнения кода JavaScript.
7. **Хранилище данных** – необходимо для сохраняемости процессов. Браузер сохраняет на жесткий диск данные различных типов, например файлы cookie. В новой спецификации HTML (HTML5) имеется определение термина "веб-база данных": это полноценная (хотя и облегченная) браузерная база данных.

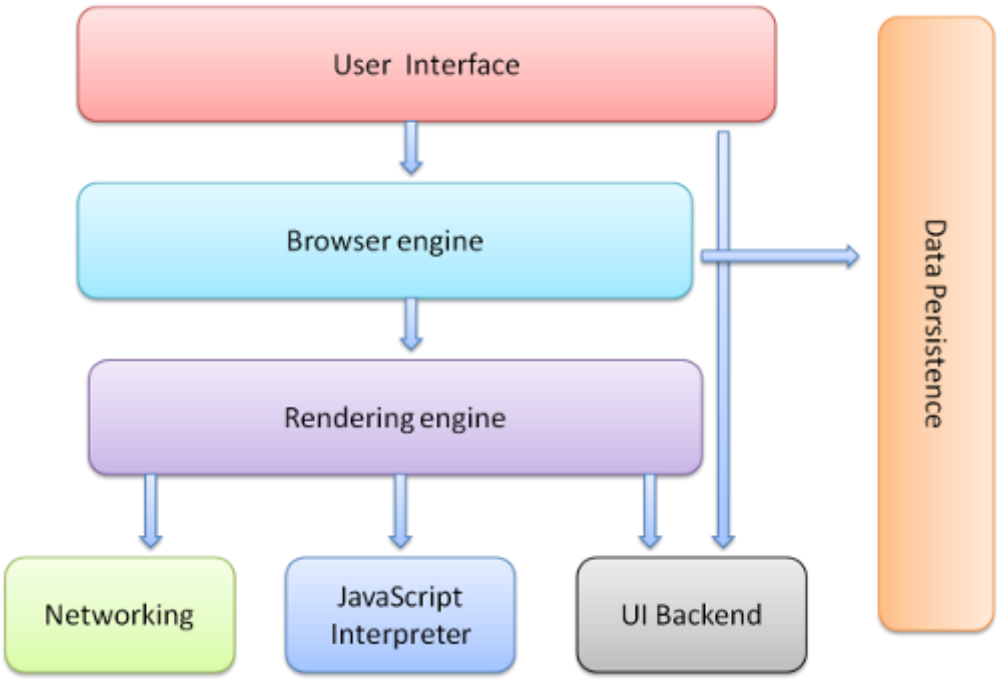


Рисунок .
Основные компоненты браузера.

Следует отметить, что Chrome, в отличие от большинства браузеров, использует несколько экземпляров модуля отображения, по одному в каждой вкладке, которые представляют собой отдельные процессы.

Модуль отображения

Как можно догадаться по названию, модуль отображения отвечает за вывод запрошенного содержания на экране браузера.

По умолчанию он способен отображать HTML- и XML-документы, а также картинки. Специальные подключаемые модули (расширения для браузеров) делают возможным отображение другого содержания, например PDF-файлов. Однако эта глава посвящена основным функциям: отображению HTML-документов и картинок, отформатированных с помощью стилей CSS.

Модули отображения

В интересующих нас браузерах (Firefox, Chrome и Safari) используются два модуля отображения. В Firefox применяется Gecko – собственная разработка Mozilla, а в Safari и Chrome используется WebKit.

WebKit представляет собой модуль отображения с открытым исходным кодом, который был изначально разработан для платформы Linux и адаптирован компанией Apple для Mac OS и Windows. Подробные сведения можно найти на сайте webkit.org.

Основная схема работы

Модуль отображения получает содержание запрошенного документа по протоколу сетевого уровня, обычно фрагментами по 8 КБ.

Схема дальнейшей работы модуля отображения выглядит приведенным ниже образом.

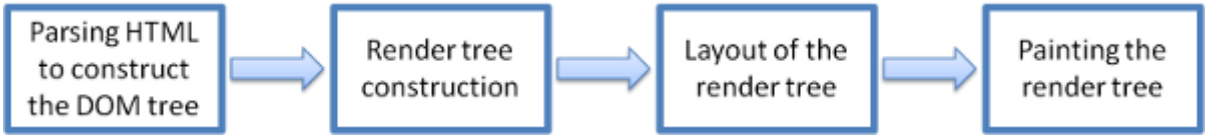


Рисунок . Схема работы модуля отображения.

Модуль отображения выполняет синтаксический анализ HTML-документа и переводит теги в узлы DOM в дереве содержания. Информация о стилях извлекается как из внешних CSS-файлов, так и из элементов style. Эта информация и инструкции по отображению в HTML-файле используются для создания еще одного дерева – дерева отображения.

Оно содержит прямоугольники с визуальными атрибутами, такими как цвет и размер. Прямоугольники располагаются в том порядке, в каком они должны быть выведены на экран.

После создания дерева отображения начинается компоновка элементов, в ходе которой каждому узлу присваиваются координаты точки на экране, где он должен появиться. Затем выполняется отрисовка, при которой узлы дерева отображения последовательно отрисовываются с помощью исполнительной части пользовательского интерфейса.

Важно понимать, что это последовательный процесс. Для удобства пользователя модуль отображения старается вывести содержание на экран как можно скорее, поэтому создание дерева отображения и компоновка могут начаться еще до завершения синтаксического анализа кода HTML. Одни части документа анализируются и выводятся на экран, в то время как другие только передаются по сети.

Примеры работы

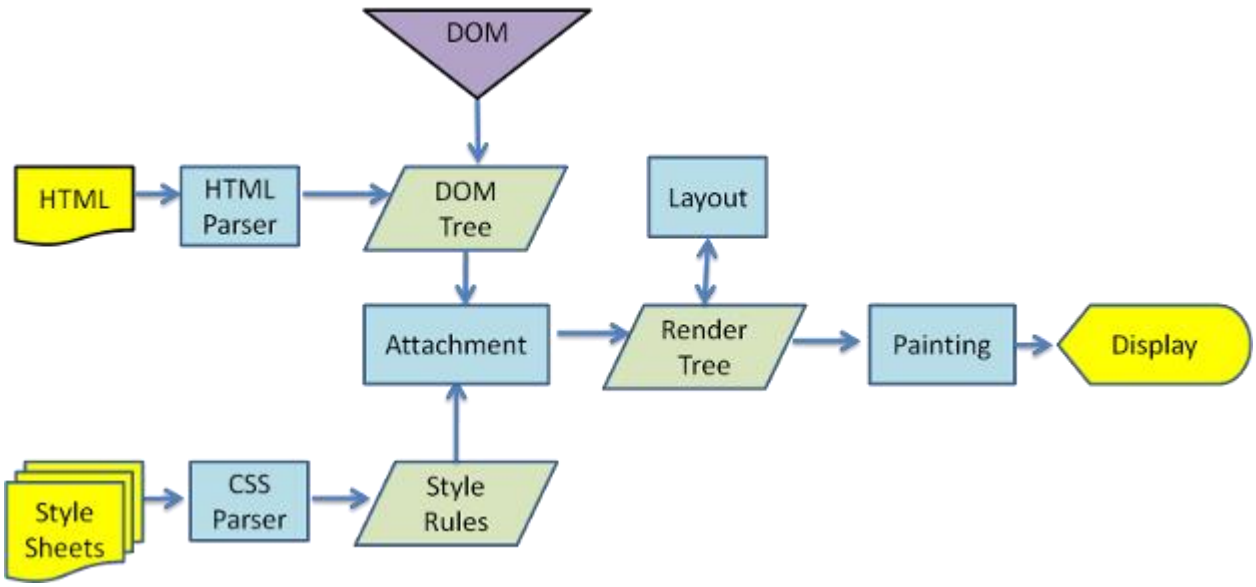


Рисунок . Схема работы модуля отображения WebKit.

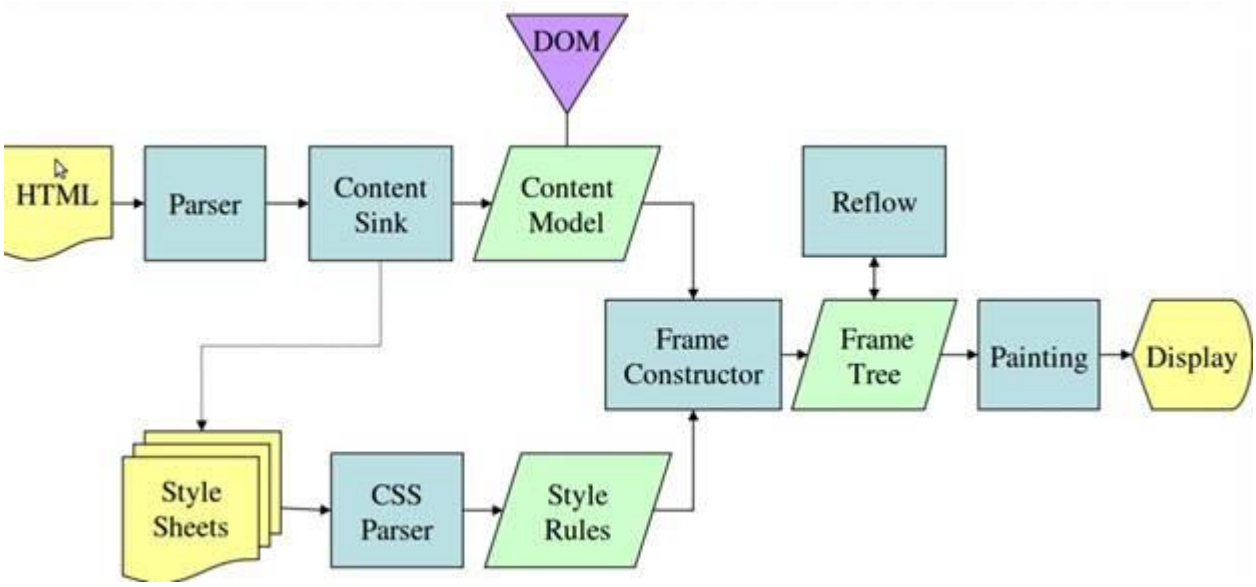


Рисунок . Схема работы модуля отображения Mozilla Gecko (3.6).

Как видно из рисунков 3 и 4, в WebKit и Gecko используется разная терминология, однако схемы их работы практически идентичны.

В Gecko дерево визуально отформатированных элементов называется деревом фреймов (frame tree), в котором каждый элемент является фреймом. В WebKit используется дерево отображения (render tree), состоящие из объектов отображения (render objects). Размещение элементов в WebKit называется компоновкой, или верстkой (layout), а в Gecko – обтеканием (reflow). Объединение узлов DOM и визуальных атрибутов для создания дерева отображения называется в WebKit совмещением (attachment). Небольшое отличие Gecko, не имеющее отношения к семантике, состоит в том, что между HTML-файлом и деревом DOM находится еще один уровень. Он называется буфером содержания (content sink) и служит для формирования элементов DOM. Теперь поговорим о каждом этапе работы подробнее.

Синтаксический анализ: общие сведения

Так как синтаксический анализ является важным этапом работы модуля отображения, рассмотрим его подробнее. Начнем с краткого введения.

Под синтаксическим анализом документа подразумевается его преобразование в пригодную для чтения и выполнения структуру. Результатом синтаксического анализа, как правило, является дерево узлов, представляющих структуру документа. Оно называется деревом синтаксического анализа, или просто синтаксическим деревом.

Например, в результате синтаксического анализа выражения 2 + 3 – 1может получиться такое дерево:

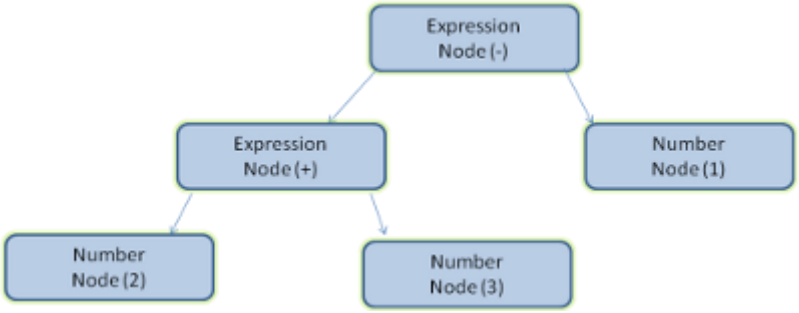


Рисунок . Узел дерева для математического выражения.

Грамматика

Синтаксический анализ работает на основе определенных правил, которые определяются языком (форматом) документа. Для каждого формата существуют грамматические правила, состоящие из словаря и синтаксиса. Они образуют т. н. бесконтекстную грамматику. Естественные языки не подчиняются правилам бесконтекстной грамматики, поэтому стандартные техники синтаксического анализа для них не годятся.

Синтаксический и лексический анализаторы

Вместе с синтаксическим применяется лексический анализ.

Лексический анализ представляет собой разделение информации на токены, или лексемы. Токены образуют словарь того или иного языка и являются конструктивными элементами для создания документов. В естественном языке токенами бы были все слова, которые можно найти в словарях.

Смысл синтаксического анализа состоит в применении синтаксических правил языка.

Анализ документа обычно выполняется двумя компонентами: **лексическим анализатором**, разбирающим входную последовательность символов на действительные токены, и **синтаксическим анализатором**, анализирующим структуру документа согласно синтаксическим правилам данного языка и формирующим синтаксическое дерево. Анализатор игнорирует неинформативные символы, такие как пробелы и переносы строк.

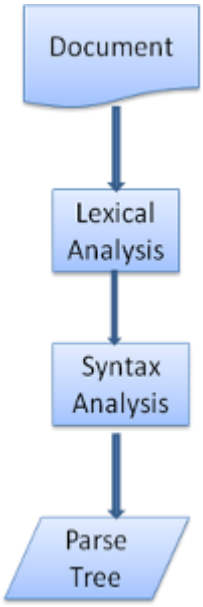


Рисунок . Переход от исходного документа к синтаксическому дереву.

Синтаксический анализ является итеративным процессом. Синтаксический анализатор обычно запрашивает у лексического новый токен и проверяет его на предмет соответствия какому-либо из синтаксических правил. Если удается установить соответствие, для токена создается новый узел в синтаксическом дереве, а анализатор запрашивает следующий токен.

Если токен не соответствует ни одному правилу, синтаксический анализатор откладывает его и запрашивает следующие токены. Так продолжается до тех пор, пока не будет найдено правило, которому бы отвечали все отложенные токены. Если найти такое правило не удается, анализатор создает исключение. Это означает, что документ содержит синтаксические ошибки и не может быть обработан полностью.

Перевод

Синтаксическое дерево не всегда бывает окончательным результатом. Синтаксический анализ часто используется в процессе перевода входного документа в нужный формат. Примером может служить компиляция. Компилятор, который переводит исходный код в машинный, сначала разбирает его и формирует синтаксическое дерево, а лишь потом создает на основе этого дерева документ с машинным кодом.

Рисунок . Этапы компиляции.

Пример синтаксического анализа

На рисунке 5 показано синтаксическое дерево, построенное на основе математического выражения. Определим элементарный математический язык и рассмотрим процесс синтаксического анализа.

Словарь: наш язык может содержать целые числа, знаки "плюс" и "минус".

Синтаксис

1. Структурными элементами языка являются выражения, операнды и операторы.
2. Язык может содержать любое количество выражений.
3. Выражение – это последовательность, состоящая из операнда, оператора и еще одного операнда.
4. Оператор – это токен "плюс" или "минус".
5. Операнд – это токен целого числа или выражение.

Рассмотрим входную последовательность символов 2 + 3 – 1.
Первый элемент, отвечающий правилу, – 2 (согласно правилу №5, это операнд). Второй такой элемент – 2 + 3 (последовательность, состоящая из операнда, оператора и еще одного операнда, определена правилом №3). Следующее соответствие мы найдем в самом конце: последовательность 2 + 3 – 1 является выражением. Так как 2+3 – это операнд, мы получаем последовательность, состоящую из операнда, оператора и еще одного операнда, что

соответствует определению выражения. Строка 2 + + не содержит соответствий правилам, поэтому была бы расценена как недействительная.

Формальное определение словаря и синтаксиса

Словарь обычно состоит из регулярных выражений.

Язык из примера выше можно было бы определить так:

INTEGER :0|[1-9][0-9]*

PLUS : +

MINUS: -

Как видите, целые числа определены регулярным выражением.

Синтаксис обычно описывается в формате BNF. Язык из примера выше можно описать так:

expression := term operation term

operation := PLUS | MINUS

term := INTEGER | expression

Как уже говорилось, язык можно обрабатывать с помощью стандартных синтаксических анализаторов, если его грамматика бесконтекстна, то есть может быть полностью выражена в формате BNF. Формальное определение бесконтекстной грамматики можно найти в этой статье Википедии.

Типы синтаксических анализаторов

Синтаксические анализаторы бывают двух типов: нисходящие и восходящие. Первые выполняют анализ сверху вниз, а вторые – снизу вверх. Нисходящие анализаторы разбирают структуру верхнего уровня и ищут соответствия синтаксическим правилам. Восходящие анализаторы сначала обрабатывают входную последовательность символов и постепенно выявляют в ней синтаксические правила, начиная с правил нижнего и заканчивая правилами верхнего уровня.

Теперь посмотрим, как эти два типа анализаторов справились бы с нашим примером.

Нисходящий анализатор начал бы с правила верхнего уровня и определил бы, что 2 + 3 –·это выражение. Затем он определил бы, что 2 + 3 – 1также является выражением (в процессе определения выражений выявляются и соответствия другим правилам, однако первым всегда рассматривается правило верхнего уровня).

Восходящий анализатор обрабатывал бы последовательность символов, пока не нашел бы подходящее правило, которым можно заменить обнаруженный фрагмент, и так до конца последовательности. Выражения с частичным соответствием при этом помещаются в стек анализатора.

Стек	**Входные символы**
	2 + 3 – 1
операнд	+ 3 – 1
оператор с операндом	3 – 1
выражение	– 1

оператор с выражением	1
выражение	

При работе такого анализатора входная последовательность символов сдвигается вправо (представьте курсор, который помещен в начало последовательности и в ходе анализа сдвигается вправо) и постепенно сводится к синтаксическим правилам.

Автоматическое создание синтаксических анализаторов

Существуют специальные приложения для создания синтаксических анализаторов, которые называются генераторами. Достаточно загрузить в генератор грамматику языка (словарный запас и синтаксические правила), и он автоматически создаст анализатор. Для создания синтаксического анализатора необходимо глубокое понимание принципов его работы, и сделать это вручную не так-то просто, поэтому генераторы бывают весьма полезны.

В WebKit используется два известных генератора: Flex для создания лексического и Bison для создания синтаксического анализатора (они также встречаются под названиями Lex и Yacc). Во Flex загружается файл с определениями токенов в регулярных выражениях, а в Bison – синтаксические правила языка в формате BNF.

Синтаксический анализатор HTML

Задача синтаксического анализатора HTML – переводить информацию из кода HTML в синтаксическое дерево.

Определение грамматики HTML

Словарь и синтаксис HTML определены в спецификациях W3C. Действующей версией является HTML4, версия HTML5 находится в разработке.

Контекстная грамматика

Как говорилось выше, синтаксические правила языка можно формально определить, например, в формате BNF.

К сожалению, ни один из описанных выше стандартных анализаторов не подходит для языка HTML (но я включила их в этот документ не просто так – они еще пригодятся, когда мы дойдем до CSS и JavaScript). HTML невозможно определить с помощью бесконтекстной грамматики, с которой работают синтаксические анализаторы.

Существует формальный стандарт определения HTML – формат DTD (Document Type Definition), однако его грамматика не является бесконтекстной.

На первый взгляд это кажется странным, ведь язык HTML не так уж далек от XML, а для XML имеется множество синтаксических анализаторов. Существует даже версия HTML на базе XML (XHTML), так в чем же разница?

Разница в том, что в HTML используется менее строгий подход: если пропущены некоторые теги (например, открывающие или закрывающие), они подставляются автоматически. Такой "мягкий" синтаксис отличается от строгого синтаксиса XML.

Это отличие кажется незначительным только на первый взгляд. С одной стороны, это основная причина популярности HTML: способность языка "прощать" ошибки ощутимо облегчает жизнь разработчику. С другой стороны, из-за этого становится сложно формально определить грамматику. Итак, грамматика HTML не является бесконтекстной, поэтому его анализ нельзя выполнить ни с помощью стандартных анализаторов, ни с помощью анализаторов XML.

DTD в HTML

Определение HTML задается в формате DTD. Он используется для формализации языков семейства SGML. Этот формат содержит определения всех допустимых элементов, их атрибутов и иерархии. Как уже упоминалось, в DTD не задается бесконтекстная грамматика.

Существует несколько версий DTD. Строгий формат в точности отвечает спецификации, а остальные также поддерживают разметку, которая использовалась браузерами в прошлом. Это необходимо для обратной совместимости с более старым содержанием. Текущую строгую версию DTD можно загрузить по адресу www.w3.org/TR/html4/strict.dtd.

DOM

Полученное синтаксическое дерево состоит из элементов DOM и узлов атрибутов. DOM – объектная модель документа (Document Object Model) – служит для представления HTML-документа и интерфейса элементов HTML таким внешним объектам, как код JavaScript.
В корне дерева находится объект Document.

Модель DOM практически идентична разметке. Рассмотрим пример разметки:

```
<html>
 <body>
  <p>
   Hello World
  </p>
  <div> <img src="example.png"/></div>
 </body>
</html>
```

Дерево DOM для этой разметки выглядит так:

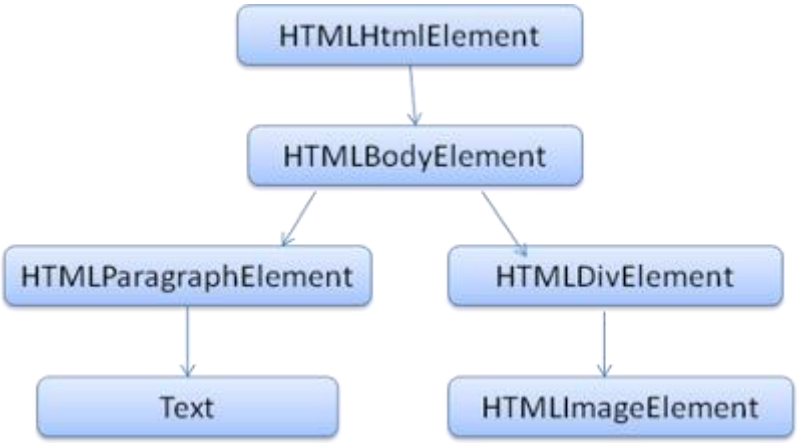


Рисунок . Дерево DOM для разметки из примера.

Как и в случае HTML, спецификации DOM разрабатывает Консорциум W3C (см. документ www.w3.org/DOM/DOMTR). Это универсальная спецификация для работы с документами. В специальном модуле описаны элементы, характерные для HTML. Определения HTML можно найти здесь: www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109/idl-definitions.html.

Под словами "дерево содержит узлы DOM" подразумевается, что дерево состоит из элементов, которые реализуют один из интерфейсов DOM. В браузерах применяются специфические реализации, обладающие дополнительными атрибутами для внутреннего использования.

Алгоритм синтаксического анализа

Как уже говорилось в предыдущих разделах, синтаксический анализ кода HTML невозможно выполнить с помощью стандартных нисходящих или восходящих анализаторов.

Ниже перечислены причины этого.

1. Язык имеет "щадящий" характер.
2. В браузерах заложены механизмы обработки некоторых частых ошибок в коде HTML.
3. Цикл синтаксического анализа характеризуется возможностью повторного вхождения. Исходный документ обычно не меняется в процессе анализа, однако в случае HTML теги скрипта, содержащие document.write, могут добавлять новые токены, поэтому исходный код может меняться.

Так как стандартные анализаторы не подходят для HTML, браузеры создают собственные анализаторы.

Алгоритм синтаксического анализа подробно описан в спецификации HTML5. Он состоит из двух этапов: лексического анализа и построения дерева.

В ходе лексического анализа входная последовательность символов разбивается на токены. К токенам HTML относятся открывающие и закрывающие теги, а также названия и значения атрибутов.

Лексический анализатор обнаруживает токен, передает его конструктору деревьев и переходит к следующему символу в поиске дальнейших токенов, и так до окончания входной последовательности.

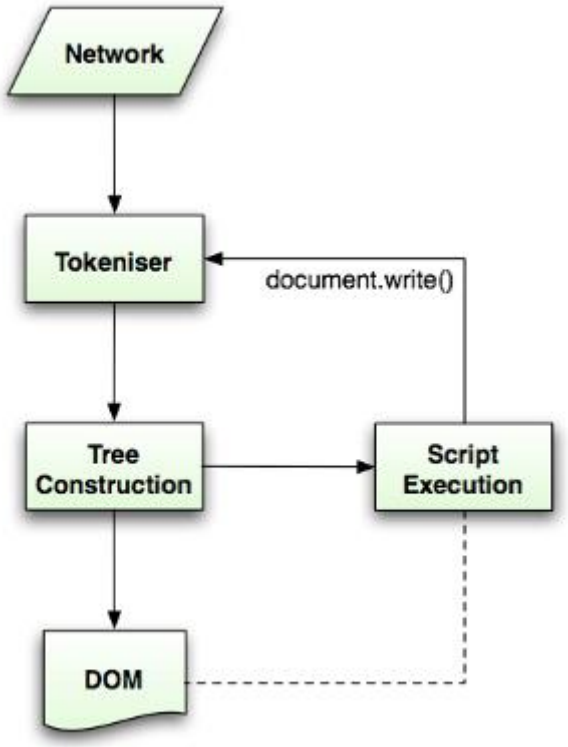


Рисунок . Этапы синтаксического анализа кода HTML (источник: спецификация HTML5).

Алгоритм лексического анализа

Результатом работы алгоритма является токен HTML. Алгоритм выражен в виде автомата с конечным числом состояний. В каждом состоянии обрабатывается один или несколько символов входной последовательности, на основе которых определяется следующее состояние. Оно зависит от этапа лексического анализа и этапа формирования дерева, то есть обработка одного и того же символа может привести к разным результатам (разным состояниям) в зависимости от текущего состояния. Алгоритм достаточно сложен, чтобы подробно описывать его здесь, поэтому рассмотрим упрощенный пример, который поможет нам лучше понять принцип его работы.

Выполним лексический анализ простого кода HTML:

```
<html>
 <body>
  Hello world
 </body>
</html>
```

Исходное состояние – "данные". Когда анализатор обнаруживает символ <, состояние меняется на **"открытый тег"**. Если далее обнаруживается буква (a–z), создается токен открывающего тега, а состояние меняется на **"название тега"**. Оно сохраняется, пока не будет обнаружен символ >. Символы по одному добавляются к названию нового токена. В нашем случае получается токен html.

При обнаружении символа > токен считается готовым и анализатор возвращается в состояние **"данные"**. Тег <body> обрабатывается точно так же. Таким образом, анализатор уже сгенерировал теги html и body и вернулся в состояние **"данные"**. Обнаружение буквы H во

фразе Hello world ведет к генерации токена символа. То же происходит с остальными буквами, пока анализатор не дойдет до символа < в теге </body>. Для каждого символа фразы Hello world создается свой токен.

Затем анализатор снова возвращается в состояние **"открытый тег"**. Обнаружение символа / ведет к созданию токена закрывающего тега и переходу в состояние **"название тега"**. Оно сохраняется, пока не будет обнаружен символ >. В этот момент генерируется токен нового тега, а анализатор снова возвращается в состояние **"данные"**. Последовательность символов </html> обрабатывается, как описано выше.

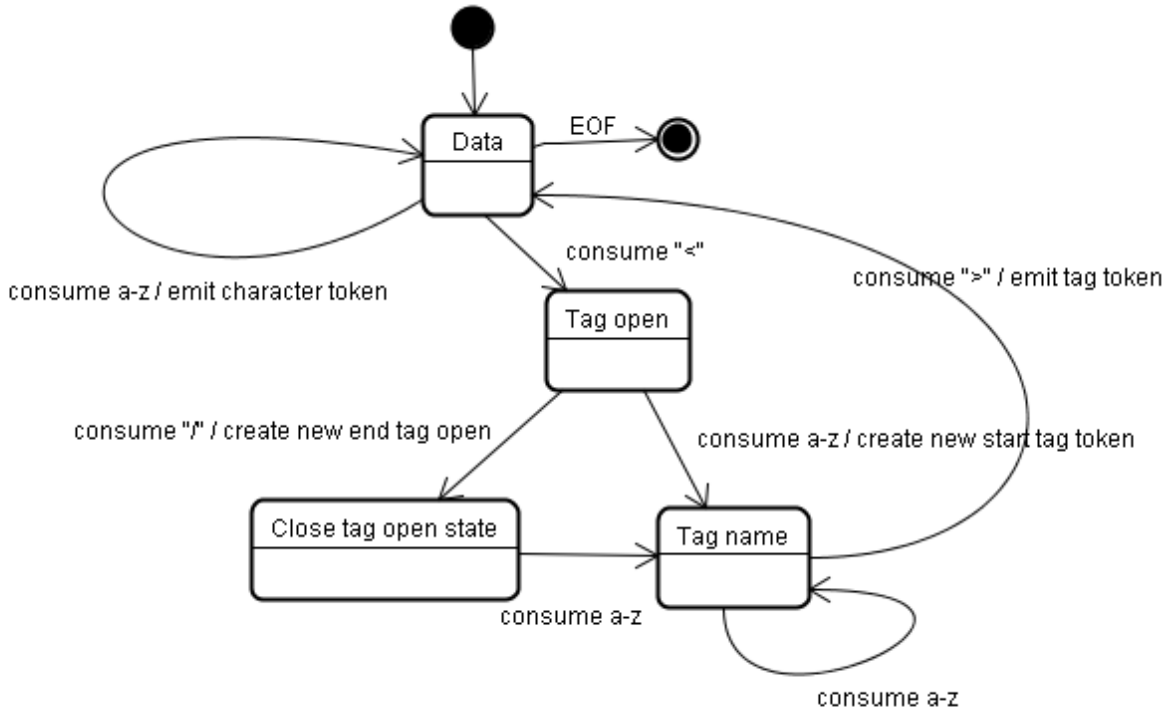


Рисунок . Лексический анализ входной последовательности символов.

Алгоритм построения дерева

При создании синтаксического анализатора формируется объект Document. На этапе построения дерево DOM, в корне которого находится этот объект, изменяется и к нему добавляются новые элементы. Каждый узел, генерируемый лексическим анализатором, обрабатывается конструктором деревьев. Для каждого токена создается свой элемент DOM, определенный спецификацией. Элементы добавляются не только в дерево DOM, но и в стек открытых элементов, который служит для исправления неправильно вложенных или незакрытых тегов. Алгоритм также выражается в виде автомата с конечным числом состояний, которые называются "способами включения" (insertion mode).

Рассмотрим этапы создания дерева для следующего фрагмента кода:

```
<html>
  <body>
    Hello world
  </body>
</html>
```

В начале этапа построения дерева у нас есть последовательность токенов, полученная в результате лексического анализа. Первое состояние называется **исходным**. При получении токена html состояние меняется на **"до html"**, после чего происходит повторная обработка токена в этом состоянии. В результате создается элемент HTMLHtmlElement, который добавляется к корневому объекту Document.

Состояние меняется на **"до head"**. Анализатор обнаруживает токен body. Хотя в нашем коде нет тега head, элемент HTMLHeadElement будет автоматически создан и добавлен в дерево.

Состояние меняется на **"внутри head"**, затем на **"после head"**. Токен body обрабатывается еще раз, создается элемент HTMLBodyElement, который добавляется в дерево, и состояние меняется на **"внутри body"**.

Теперь пришла очередь токенов строки Hello world. Обнаружение первого из них ведет к созданию и вставке узла Text, к которому затем добавляются остальные символы.

При получении закрывающего токена body состояние меняется на **"после body"**. Когда анализатор доходит до закрывающего тега html, состояние меняется на **"после после body"**. При получении токена конца файла анализ завершается.

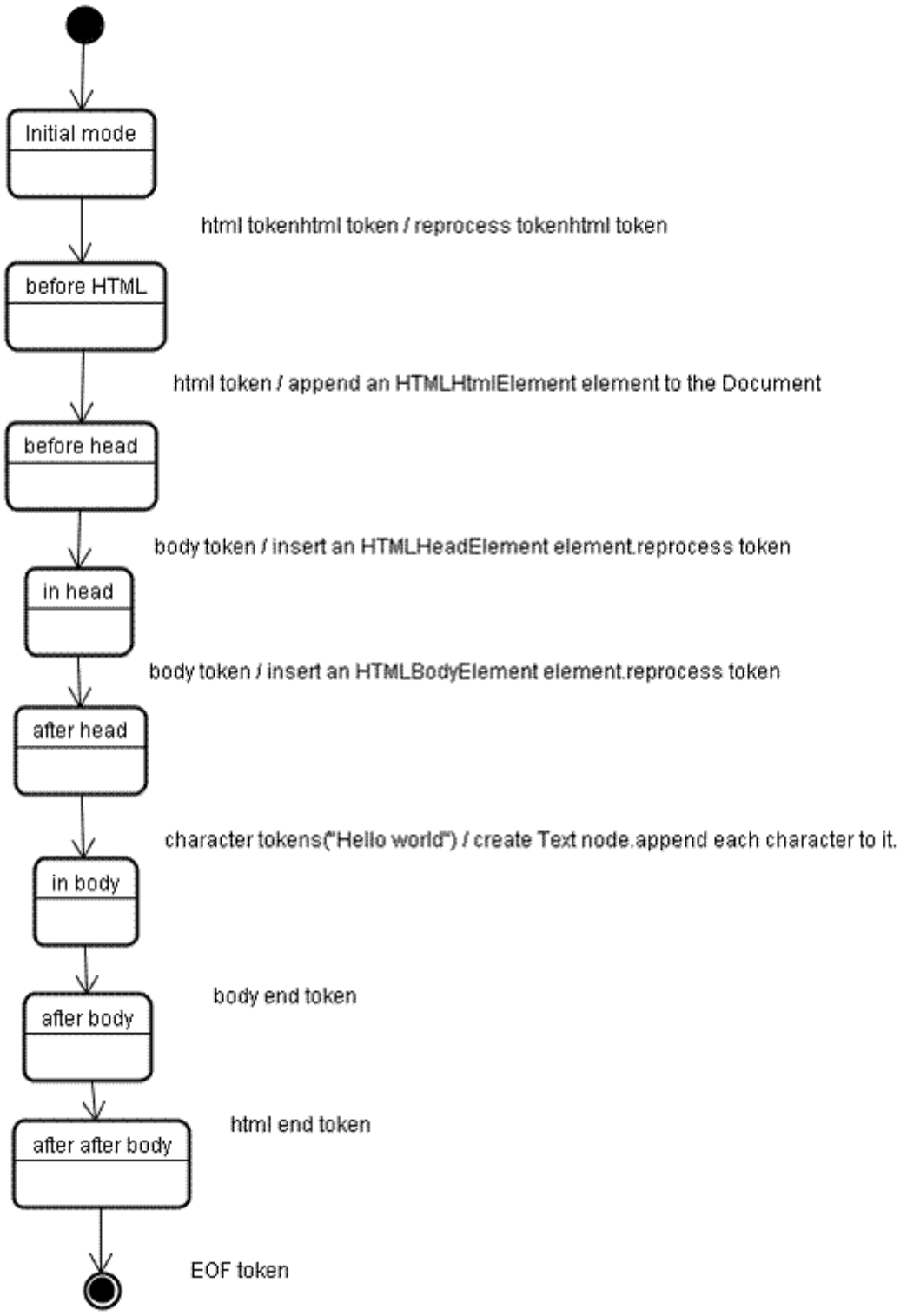


Рисунок .

Построение дерева для кода HTML из примера.

Действия после синтаксического анализа

На этом этапе браузер помечает документ как интерактивный и начинает анализ отложенных скриптов, которые необходимо выполнить после завершения анализа документа. Состояние документа затем меняется на "готово", и вызывается событие load.

В спецификации HTML5 подробно описаны алгоритмы лексического анализа и построения деревьев.

Обработка ошибок браузерами

На странице HTML вы никогда не увидите ошибку "Недопустимый синтаксис". Браузеры умеют корректировать ошибки содержания, не прерывая работу.

Рассмотрим вот такой код HTML:

```
<html>
 <mytag>
 </mytag>
 <div>
 <p>
 </div>
  Really lousy HTML
 </p>
</html>
```

В этом коротком фрагменте я нарушила множество правил (mytag не является стандартным тегом, теги p и div вложены неверно и т. д.), однако браузер не испытывает никаких проблем и отображает содержание корректно. Большая часть кода синтаксического анализатора служит для исправления ошибок разработчиков.

В браузерах используются очень похожие механизмы обработки ошибок, но, как ни странно, они не описаны в текущей спецификации HTML. Как и закладки или кнопки навигации, они просто появились в результате многолетней эволюции браузеров. Существуют недопустимые конструкции HTML, которые довольно часто встречаются на сайтах, и разные браузеры исправляют их похожими способами.

В спецификации HTML5 определены некоторые требования к этому механизму. В WebKit они указаны в комментарии к классу parser.

Синтаксический анализатор обрабатывает входящие токены и создает дерево документа. Если документ написан без ошибок, выполняется его стандартный анализ.

К сожалению, многие документы HTML содержат ошибки, и синтаксический анализатор должен быть к ним готов.

Он должен уметь обрабатывать как минимум перечисленные ниже типы ошибок.

1. *Использование добавляемого элемента явно запрещено одним из внешних тегов. В этом случае необходимо закрыть все теги, кроме того, который запрещает использование данного элемента, и добавить этот элемент в самом конце.*
2. *Элемент нельзя добавить напрямую. Возможно, автор документа забыл вставить тег между элементами (или такой тег необязателен). Это касается тегов HTML, HEAD, BODY, TBODY, TR, TD, LI (надеюсь, я ничего не забыла).*
3. *Блочный элемент добавлен внутрь строчного. Необходимо закрыть все строчные элементы вплоть до следующего в иерархии блочного элемента.*
4. *Если это не помогает, необходимо закрывать элементы, пока не появится возможность добавить нужный элемент или проигнорировать тег.*

Рассмотрим примеры того, как WebKit обрабатывает некоторые ошибки.

**Тег </br> вместо
**

На некоторых сайтах можно встретить тег </br> там, где должен быть
. Чтобы отобразить содержание в браузерах IE и Firefox, WebKit обрабатывает этот тег как
.
Код:

```
if (t->isCloseTag(brTag) && m_document->inCompatMode()) {
    reportError(MalformedBRError);
    t->beginTag = true;
}
```

Обратите внимание: механизм обработки ошибок является встроенным, пользователь его не видит.

"Заблудившаяся" таблица

"Заблудившейся" называется таблица, которая находится внутри другой таблицы, но не внутри одной из ее ячеек.
Пример кода:

```
<table>
  <table>
    <tr><td>inner table</td></tr>
  </table>
  <tr><td>outer table</td></tr>
</table>
```

WebKit меняет иерархию, превращая таблицы в элементы одного уровня:

```
<table>
  <tr><td>outer table</td></tr>
</table>
<table>
  <tr><td>inner table</td></tr>
</table>
```

Код:

```
if (m_inStrayTableContent && localName == tableTag)
      popBlock(tableTag);
```

В системе отображения WebKit используется стек для хранения текущих элементов, который выводит внутреннюю таблицу из стека внешней. Теперь таблицы находятся на одном уровне иерархии.

Вложенные формы

Если автор поместит один элемент form внутрь другого такого элемента, последний будет игнорироваться.
Код:

```
if (!m_currentFormElement) {
    m_currentFormElement = new HTMLFormElement(formTag,    m_document);
}
```

Слишком много вложенных тегов

Комментарий ниже говорит сам за себя.

На сайте www.liceo.edu.mx иерархия тегов имеет около 1500 уровней, и все это теги <b>. Разрешено использовать не более 20 вложенных тегов одного типа, в противном случае все они будут игнорироваться.

```
bool HTMLParser::allowNestedRedundantTag(const AtomicString& tagName)
{

unsigned i = 0;
for (HTMLStackElem* curr = m_blockStack;
     i < cMaxRedundantTagDepth && curr && curr->tagName == tagName;
   curr = curr->next, i++) { }
return i != cMaxRedundantTagDepth;
}
```

Неправильное размещение закрывающих тегов html или body

И снова комментарий говорит сам за себя.

Поддержка по-настоящему неграмотных документов HTML: никогда не закрываем тег body, так как некоторые особо талантливые разработчики пытаются закрыть его раньше, чем кончается документ. Чтобы закрыть теги, используем метод end().

```
if (t->tagName == htmlTag || t->tagName == bodyTag )
    return;
```

Веб-разработчикам на заметку: если вы не хотите прославиться в подобных комментариях к механизму обработки ошибок WebKit, пишите качественный код HTML.

Синтаксический анализ CSS

Помните, как в начале учебника мы рассматривали подходы к синтаксическому анализу? В отличие от HTML, в CSS используется бесконтекстная грамматика, поэтому для анализа подходят стандартные средства, о которых мы уже говорили. Кроме того, лексические и синтаксические правила CSS определены в спецификации CSS.

Рассмотрим несколько примеров.
Лексическая грамматика (словарь) определяется регулярными выражениями для каждого токена:

```
comment   \/\*[^*]*\*+([^/*][^*]*\*+)*\/

num   [0-9]+|[0-9]*"."[0-9]+

nonascii  [\200-\377]

nmstart   [_a-z]|{nonascii}|{escape}

nmchar    [_a-z0-9-]|{nonascii}|{escape}

name    {nmchar}+

ident   {nmstart}{nmchar}*
```

Ident – это идентификатор, который используется как название класса. Name – это элемент id, для ссылки на него используется символ решетки (#).

Синтаксические правила описаны в формате BNF.

```
ruleset

  : selector [ ',' S* selector ]*

    '{' S* declaration [ ';' S* declaration ]* '}' S*

  ;

selector

  : simple_selector [ combinator selector | S+ [ combinator? selector ]? ]?

  ;

simple_selector

  : element_name [ HASH | class | attrib | pseudo ]*

  | [ HASH | class | attrib | pseudo ]+

  ;

class

  : '.' IDENT

  ;

element_name

  : IDENT | '*'

  ;

attrib

  : '[' S* IDENT S* [ [ '=' | INCLUDES | DASHMATCH ] S*

    [ IDENT | STRING ] S* ] ']'

  ;

pseudo

  : ':' [ IDENT | FUNCTION S* [IDENT S*] ')' ]
```

;

Набор правил (ruleset) представляет собой описанную ниже структуру.

```
div.error , a.error {
  color:red;
  font-weight:bold;
}
```

Элементы div.error и a.error – это селекторы. Действующие правила данного набора заключены в фигурные скобки. Формально эта структура определяется так:

```
ruleset
  : selector [ ',' S* selector ]*
    '{' S* declaration [ ';' S* declaration ]* '}' S*
  ;
```

Это означает, что набор правил действует как селектор или как несколько селекторов, разделенных запятыми и пробелами (S означает пробел). Набор правил содержит одно или несколько объявлений, разделенных точкой с запятой. Они заключены в фигурные скобки. Определения понятий "объявление" и "селектор" будут даны ниже.

Синтаксический анализатор CSS в WebKit

В WebKit для автоматического создания синтаксических анализаторов CSS используются генераторы Flex и Bison. Как уже говорилось, Bison служит для создания восходящих анализаторов, при работе которых входная последовательность символов сдвигается вправо. В Firefox используется нисходящий анализатор, разработанный организацией Mozilla. В обоих случаях файл CSS разбирается на объекты StyleSheet, содержащие правила CSS. Объект правил CSS содержит селектор и объявление, а также другие объекты, характерные для грамматики CSS.

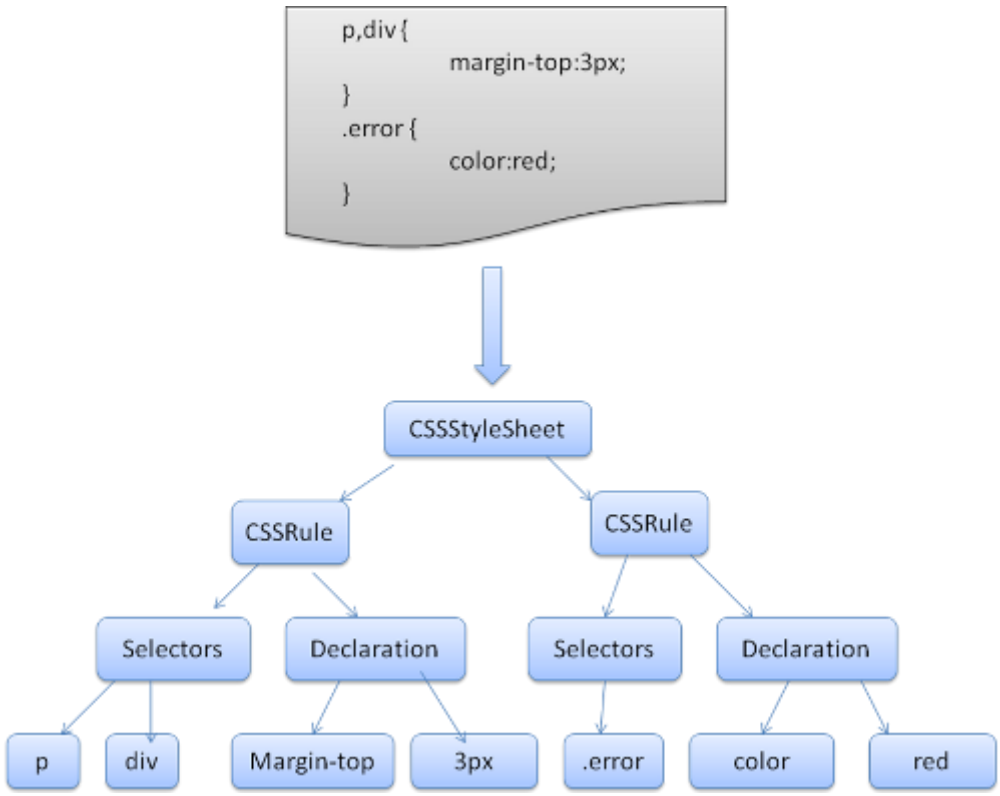


Рисунок . Синтаксический анализ CSS.

Порядок обработки скриптов и таблиц стилей

Скрипты

Веб-документы придерживаются синхронной модели. Предполагается, что скрипты будут анализироваться и исполняться сразу же, как только анализатор обнаружит тег <script>. Синтаксический анализ документа откладывается до завершения выполнения скрипта. Если речь идет о внешнем скрипте, сначала необходимо запросить сетевые ресурсы. Это также делается синхронно, а анализ откладывается до получения ресурсов. Такая модель использовалась много лет и даже занесена в спецификации HTML 4 и 5. Разработчик мог пометить скрипт тегом defer, чтобы синтаксический анализ документа можно было выполнять до завершения выполнения скрипта. В HTML5 появилась возможность пометить скрипт как асинхронный (asynchronous), чтобы он анализировался и выполнялся в другом потоке.

Ориентировочный синтаксический анализ

Этот механизм оптимизации используется и в WebKit, и в Firefox. При выполнении скриптов остальные части документа анализируются в другом потоке, чтобы оценить необходимые ресурсы и загрузить их из сети. Таким образом, ресурсы загружаются в параллельных потоках, что повышает общую скорость обработки. Обратите внимание: ориентировочный анализатор не изменяет дерево DOM (это работа основного анализатора), а лишь обрабатывает ссылки на внешние ресурсы, такие как внешние скрипты, таблицы стилей и картинки.

Таблицы стилей

Таблицы стилей основаны на другой модели. Так как они не вносят изменений в дерево DOM, теоретически останавливать анализ документа, чтобы дождаться их обработки, бессмысленно. Однако скрипты могут запрашивать данные о стилях на этапе синтаксического анализа документа. Если стиль еще не загружен и не проанализирован, скрипт может получить неверную информацию. Разумеется, это повлекло бы за собой целый ряд проблем. Если Firefox

обнаруживает таблицу стилей, которая еще не загружена и не проанализирована, то все скрипты останавливаются. В WebKit они останавливаются только в случае, если пытаются извлечь свойства стилей, которые могут быть определены в незагруженных таблицах.

Построение дерева отображения

Во время построения дерева DOM браузер создает еще одну структуру – дерево отображения. В нем визуальные элементы размещаются в том порядке, в каком их необходимо вывести на экран. Это визуальное представление документа. Дерево отображения служит для того, чтобы отрисовка содержания выполнялась в правильном порядке.

В Firefox элемент дерева отображения называется "фреймом" (frame). В WebKit используется термин "объект отображения" (render object).
Каждый объект отображения располагает данными об отрисовке самого себя и своих дочерних элементов.
Класс RenderObject – основной класс объектов отображения в WebKit – определен следующим образом:

```
class RenderObject{

  virtual void layout();

  virtual void paint(PaintInfo);

  virtual void rect repaintRect();

  Node* node;  //the DOM node

  RenderStyle* style;  // the computed style

  RenderLayer* containgLayer; //the containing z-index layer

}
```

Каждый объект отображения представляет собой прямоугольную область, соответствующую окну CSS узла, как описано в спецификации CSS2. Он содержит геометрические данные, такие как ширина, высота и положение.
Тип окна зависит от атрибута display объекта style, назначенного данному узлу (см. раздел Вычисление стилей). Ниже представлен код, который используется в WebKit, чтобы определить, какой тип объекта отображения необходимо создать для узла DOM, на основе атрибута свойства display.

```
RenderObject* RenderObject::createObject(Node* node, RenderStyle* style)

{

   Document* doc = node->document();

   RenderArena* arena = doc->renderArena();

   ...

   RenderObject* o = 0;


   switch (style->display()) {

      case NONE:
```

```
        break;
      case INLINE:
        o = new (arena) RenderInline(node);
        break;
      case BLOCK:
        o = new (arena) RenderBlock(node);
        break;
      case INLINE_BLOCK:
        o = new (arena) RenderBlock(node);
        break;
      case LIST_ITEM:
        o = new (arena) RenderListItem(node);
        break;
     ...
  }

  return o;
}
```

Учитывается и тип элемента: например, для элементов управления формами и таблиц используются специальные фреймы.
В WebKit, если элемент пытается создать специальный объект отображения, метод createRenderer будет переопределен. Объекты отображения указывают на объекты style, содержащие негеометрическую информацию.

Как дерево отображения связано с деревом DOM

Объекты обработки соответствуют элементам DOM, но не идентичны им. Невизуальные элементы DOM не включаются в дерево отображения (примером может служить элемент head). Кроме того, в дерево не включаются элементы, у которых для свойства display задан атрибут none (элементы с атрибутом hidden включаются).

Существуют и такие элементы DOM, которым соответствует сразу несколько визуальных объектов. Обычно это элементы со сложной структурой, которые невозможно описать одним-единственным прямоугольником. Например, элементу select соответствуют три визуальных объекта: один для области отображения, другой для раскрывающегося списка, третий для кнопки. Кроме того, если текст не вмещается на одну строку и разбивается на фрагменты, новые строки добавляются как самостоятельные объекты отображения.
Еще одним примером, где используется несколько объектов отображения, является некорректно написанный код HTML. Согласно спецификации CSS, строчный элемент может содержать либо только блочные, либо только строчные элементы. Если же содержание смешанное, то в качестве оболочки для строчных объектов создаются анонимные блочные объекты.

Некоторым объектам отображения соответствует узел DOM, но их положения в дереве не совпадают. Плавающие элементы и элементы с абсолютными координатами исключаются из общего процесса, помещаются в отдельную часть дерева и затем отображаются в стандартном фрейме, хотя на самом деле должны отображаться во фрейме-заполнителе.

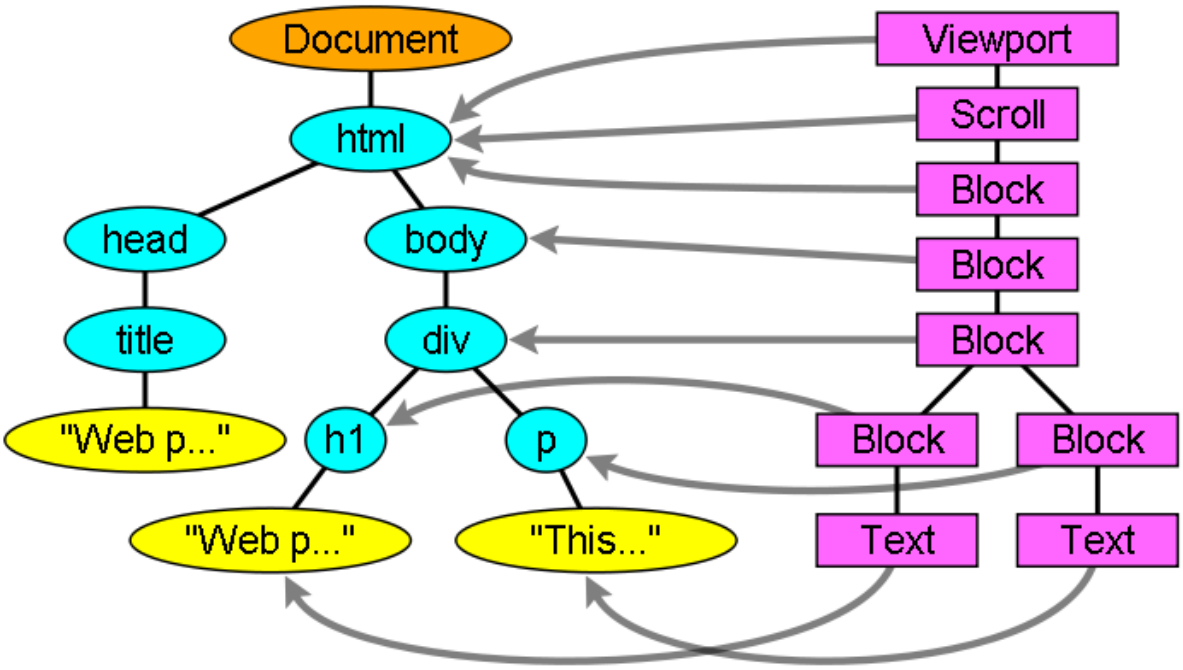


Рисунок . Дерево отображения и соответствующее ему дерево DOM (3.1). Viewport (область просмотра) – это главный контейнер. В WebKit он представлен объектом RenderView.

Процесс построения дерева

В Firefox визуальное представление регистрируется как слушатель обновлений DOM. Создание фреймов делегируется конструктору FrameConstructor, который определяет стили (см. Вычисление стилей) и создает фрейм.

В WebKit процесс определения стиля и создания объекта отображения называется совмещением (attachment). Каждый узел DOM имеет метод attach. Совмещение выполняется синхронно; при добавлении нового узла в дерево DOM для него вызывается метод attach.

В результате обработки тегов html и body создается корневой объект дерева отображения. В спецификации CSS он называется контейнером – блоком верхнего уровня, в котором содержатся все остальные блоки. Его размеры формируют область просмотра, то есть часть окна браузера, в которой будет показано содержание. В Firefox она называется ViewPortFrame, а в WebKit – RenderView. Это объект отображения, на который указывает документ. Остальное дерево строится посредством добавления в него узлов DOM.

Подробные сведения о модели обработки приведены в спецификации CSS2.

Вычисление стилей

Чтобы построить дерево отображения, необходимо рассчитать визуальные свойства каждого объекта. Для этого вычисляются свойства стиля каждого элемента.

Стиль определяется различными таблицами стилей, строчными элементами style и визуальными свойствами в документе HTML (такими как bgcolor). Последние переводятся в свойства CSS.

Таблицы стилей могут быть предоставлены браузером, разработчиком веб-страницы или пользователем, который может выбрать в браузере предпочитаемый стиль (например, в Firefox это можно сделать, поместив таблицу стилей в папку Firefox Profile).

С вычислением стилей связан ряд сложностей.

1. Данные стилей содержат множество свойств и бывают очень объемны, что может вести к проблемам с памятью.
2. Поиск подходящих правил для каждого элемента может замедлить работу, если код не оптимизирован. Если подставлять к каждому элементу все правила по очереди, это заметно отразится на производительности. Селекторы могут иметь сложную структуру, поэтому даже если определенная последовательность правил сначала покажется подходящей, в ходе анализа может оказаться, что это не так, и придется пробовать другой вариант.

Вот пример сложного селектора:

```
div div div div{

  ...

}
```

В данном случае правила необходимо применить к элементу , являющемуся потомком трех других элементов div. Предположим, нам требуется проверить, подходит ли определенное правило для данного элемента . Мы выбираем в дереве некую последовательность правил для проверки. Может оказаться, что мы проверим почти целое дерево, а в итоге обнаружим, что элементов div всего два и, следовательно, правило не применимо. Придется пробовать другую последовательность.

3. Применение правил подразумевает определение иерархии для достаточно сложных перекрывающихся правил.

Как браузеры справляются с этой задачей?

Совместное использование информации о стилях

Узлы в WebKit соответствуют объектам RenderStyle, которые в ряде случаев могут одновременно использоваться и другими узлами. Это возможно, если узлы расположены на одном уровне и соблюдены все перечисленные ниже условия.

1. Все элементы отвечают одному состоянию мыши (например, :hover).
2. Ни для одного из элементов не прописано свойство id.
3. Названия тегов одинаковы.
4. Атрибуты классов одинаковы.
5. Отображаемые атрибуты одинаковы.
6. Состояния ссылок одинаковы.
7. Состояния фокуса ввода одинаковы.
8. Ни для каких элементов селекторы не совпадают таким образом, чтобы использовался селектор атрибута, находящийся в любом месте внутри селектора.

9. Для элементов не заданы атрибуты строчных объектов style.
10. Отсутствуют селекторы того же уровня. WebCore при обнаружении селектора того же уровня вызывает оператор switch и запрещает совместное использование стилей во всем документе. Это относится к селекторам +, :first-child и :last-child.

Дерево правил Firefox

Чтобы упростить вычисление стилей, в Firefox используются две дополнительных структуры: дерево правил и дерево контекстов стилей. В WebKit также есть объекты стилей, однако они не сохраняются в специальном дереве. Вместо этого узлы DOM указывают на соответствующие стили.

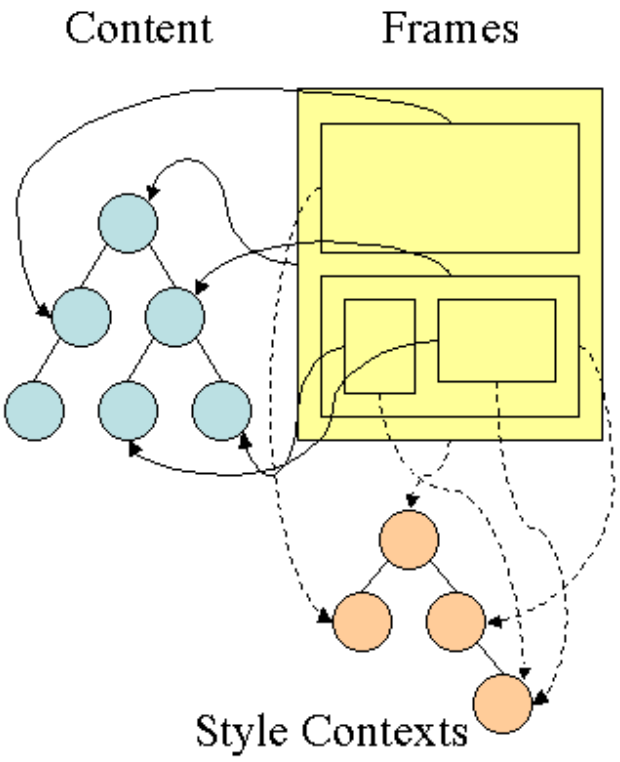
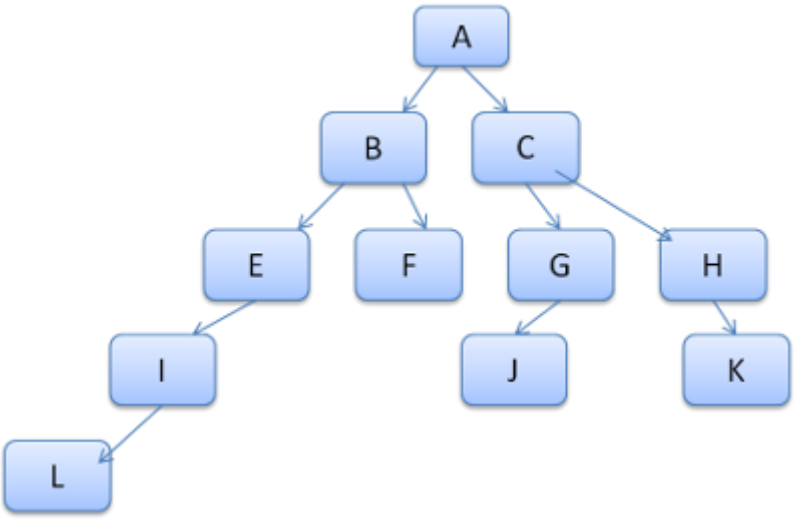


Рисунок . Дерево контекстов стилей Firefox (2.2).

В контекстах стилей содержатся их конечные значения. Они вычисляются путем применения всех соответствующих правил в нужном порядке и перевода логических значений в абсолютные. Например, если логическое значение задано в процентах от ширины экрана, оно будет переведено в абсолютные единицы. Благодаря дереву правил узлы могут совместно использовать уже рассчитанные значения, а не вычислять их снова. Кроме того, это экономит место.

Все подходящие правила сохраняются в дереве. Узлы, расположенные в нижней части последовательности, имеют более высокий приоритет. Дерево содержит все обнаруженные последовательности подходящих правил. Их сохранение не требует особых ресурсов. Дерево не рассчитывается для каждого узла. Вместо этого, если требуется вычислить данные о стиле узла, к дереву добавляются уже обработанные последовательности.

Последовательности правил в дереве напоминают слова лексикона. Предположим, мы уже рассчитали вот такое дерево правил:



Далее представим, что нам требуется подобрать правила для другого элемента в дереве контента и применить их в порядке B – E – I. Эта последовательность уже присутствует в дереве, так как ранее мы рассчитывали последовательность A – B – E – I – L. Это значительно упрощает нашу текущую задачу.

Рассмотрим подробнее, за счет чего это происходит.

Разделение на структуры

Контексты стилей делятся на структуры. Структуры содержат информацию о стилях для той или иной категории (например, для цвета или рамки). Свойства структуры могут быть наследуемыми или ненаследуемыми. Наследуемыми называются свойства, полученные от родительского элемента (кроме случаев, когда они определены элементом самостоятельно). Ненаследуемые свойства используют значения по умолчанию (если они не определены).

В дереве сохраняются целые структуры (включая уже вычисленные конечные значения). Смысл в том, что если узел нижнего уровня не определяет структуру, можно воспользоваться кэшированной структурой, сохраненной в одном из узлов верхних уровней.

Расчет контекстов стилей с помощью дерева правил

Прежде чем вычислять контексты стилей для отдельных элементов, необходимо рассчитать в дереве правил новую последовательность (или воспользоваться одной из существующих). После этого можно применить правила из последовательности, чтобы заполнить структуры в новом контексте стиля. Начнем с нижнего узла, так как он имеет самый высокий приоритет (обычно это наиболее специфичный селектор), и будем двигаться по дереву вверх, пока не заполним структуру. Если этот узел не позволяет определить структуру, у нас есть возможность оптимизировать задачу. Мы будем двигаться по дереву вверх, пока не найдем узел, который полностью определяет ее, и установим ссылку на него. Это идеальное решение: так узлы могут совместно использовать всю структуру целиком, а нам не нужно выделять память для повторного расчета конечных значений.
Если мы обнаружим частичное определение, то будем двигаться по дереву вверх, пока не заполним всю структуру.

Если определение для структуры найти не удалось, существует два варианта. Если структура наследуемая, мы установим ссылку на структуру родительского элемента в **дереве контекстов стилей**. В этом случае речь также идет о совместном использовании структур. Если же структура ненаследуемая, используются значения по умолчанию.

Если самый специфичный узел содержит какие-либо значения, выполним некоторые дополнительные вычисления, чтобы получить абсолютные единицы. Результат затем сохраняется в узле дерева, чтобы его могли использовать дочерние элементы.

Если имеется элемент того же уровня, указывающий на этот узел, контекст стиля может быть предоставлен ему **целиком**.

Рассмотрим следующий код HTML:

```
<html>
 <body>
  <div class="err" id="div1">
   <p>
    this is a <span class="big"> big error </span>
    this is also a
    <span class="big"> very  big  error</span> error
   </p>
  </div>
  <div class="err" id="div2">another error</div>
 </body>
</html>
```

Определим следующие правила:

1. div {margin:5px;color:black}
2. .err {color:red}
3. .big {margin-top:3px}
4. div span {margin-bottom:4px}
5. #div1 {color:blue}
6. #div2 {color:green}

Чтобы не усложнять задачу, предположим, что нам требуется заполнить только две структуры: цвета (color) и поля (margin). Структура color содержит только один член – цвет, а структура margin содержит четыре стороны.
Полученное дерево выглядит так (надписи на узлах состоят из названия узла и номера правила, на которое они указывают):

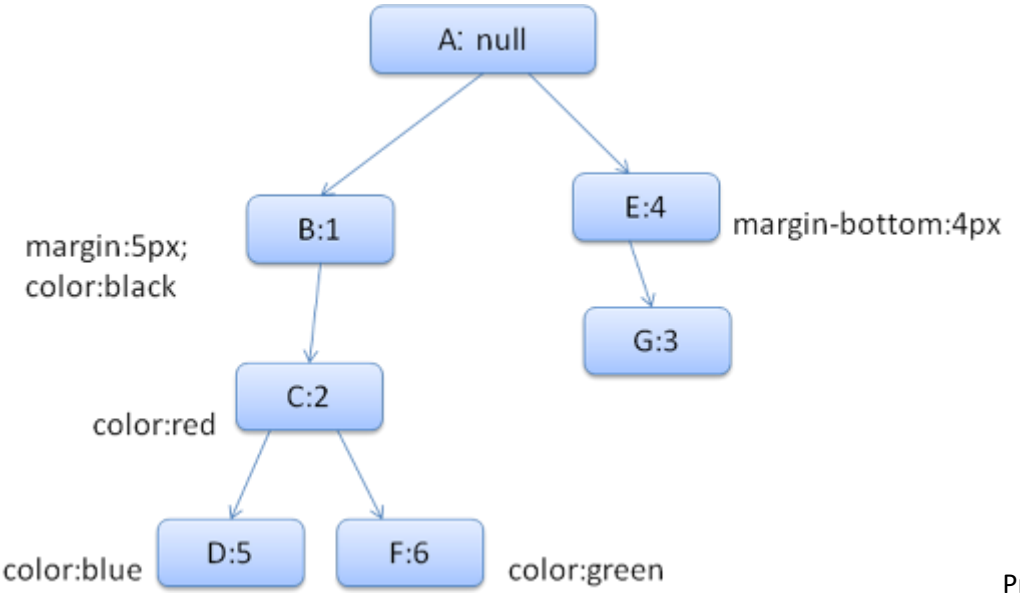


Рисунок . Дерево правил.

Дерево контекстов стилей выглядит так (надписи на узлах состоят из названия узла и номера правила, на которое они указывают):

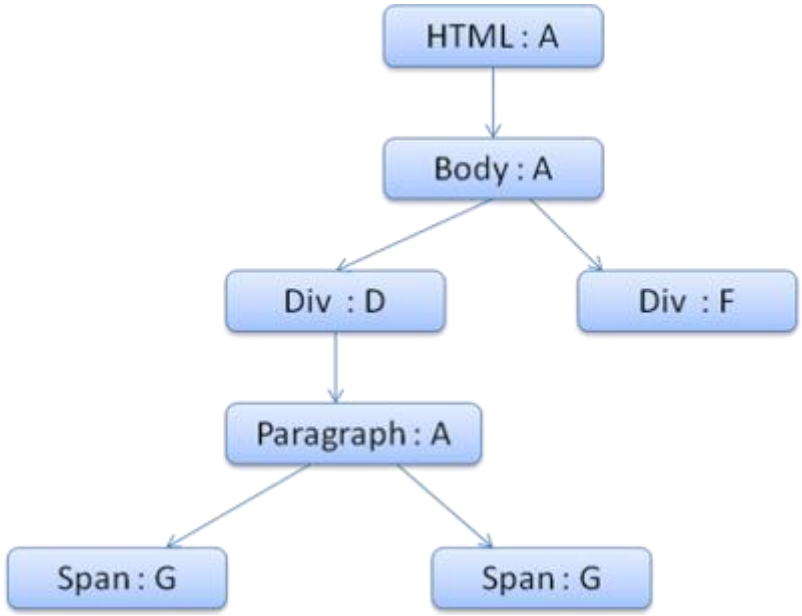


Рисунок . Дерево контекстов стилей.

Предположим, мы выполняем синтаксический анализ кода HTML и добрались до второго тега . Нам требуется создать контекст стиля для этого узла и заполнить его структуры.
Путем сопоставления правил мы обнаружим, что для тега подходят правила 1, 2 и 6. Это означает, что в дереве уже присутствует последовательность, которую можно использовать, а нам остается лишь добавить еще один узел для правила 6 (узел F в дереве правил).
Так мы создали контекст стиля и поместили его в дерево. Новый контекст указывает на узел F в дереве правил.

Теперь заполним стилевые структуры. Начнем со структуры margin. Так как последний узел (F) не определяет ее значение, мы будем двигаться по дереву вверх, пока не обнаружим кэшированную структуру, рассчитанную при добавлении предыдущего узла. Нужная структура расположена в узле B: это узел самого верхнего уровня, в котором определяются правила отрисовки полей.

У нас уже есть определение для структуры color, поэтому мы не можем использовать кэшированную структуру. Так как структура color имеет всего один атрибут, нам не требуется

искать остальные на верхних уровнях дерева. Мы вычислим конечное значение (переведем строку в формат RGB и т. д.) и сохраним рассчитанную структуру в этом узле.

Работать со вторым элементом еще проще. Мы сопоставим его с правилами и обнаружим, что он указывает на правило G, как и предыдущий элемент span. Так как два элемента одинакового уровня указывают на один и тот же узел, они могут одновременно использовать целый контекст стиля: достаточно установить ссылку на контекст предыдущего элемента span.

Если структура содержит правила, унаследованные от родительского элемента, в дереве контекстов выполняется кэширование (свойство color является наследуемым, однако Firefox обрабатывает его как ненаследуемое и сохраняет в дереве правил).
Добавим в абзац правила отображения шрифтов:

p {font-family:Verdana;font size:10px;font-weight:bold}

Элемент p, являющийся дочерним по отношению к div в дереве контекстов, может использовать для определения шрифтов соответствующую структуру своего родителя, если для этого элемента p не заданы собственные правила отображения шрифтов.

В WebKit, где нет дерева правил, совпадающие объявления обрабатываются четыре раза. Сначала применяются свойства с высоким приоритетом без пометки !important (те, которые необходимо применить в начале, так как от них зависят остальные свойства, такие как display), затем свойства с высоким приоритетом и с пометкой !important, затем свойства с обычным приоритетом без пометки !important, и, наконец, свойства с обычным приоритетом и с пометкой !important. Таким образом, если свойство вычисляется несколько раз, конечный результат выбирается согласно приоритету. Последняя группа имеет наивысший приоритет.

Итак, возможность совместного использования объектов style (целиком или только отдельных структур) решает проблемы №1 и №3. Дерево правил Firefox также позволяет применять свойства в нужном порядке.

Классификация правил для упрощения сопоставления

Правила стилей извлекаются из нескольких источников.

- Правила CSS, определенные во внешних таблицах стилей или в элементах style:

p {color:blue}

- Атрибуты строчных объектов style:

<p style="color:blue" />

- Визуальные атрибуты HTML (сопоставляются с подходящими правилами стилей):

<p bgcolor="blue" />

Последние два источника легко сопоставить с элементом, так как он содержит атрибуты объекта style и при сопоставлении атрибутов HTML может служить ключом.

Как уже упоминалось (см. проблему №2), сопоставление правил CSS не так однозначно. Чтобы упростить задачу, правила классифицируются.

После синтаксического анализа таблицы стилей каждое правило добавляется в одну или в несколько хэш-карт в зависимости от селектора. Существуют карты, организованные по идентификатору, названию класса или тега, а также общие карты для всех остальных случаев. Если селектором является идентификатор, правило добавляется в карту идентификаторов, если класс,

то в карту классов и т. д.
Такая классификация упрощает поиск подходящих правил. Нам не приходится проверять все объявления: достаточно извлечь из карты подходящие правила. Такая классификации позволяет сразу отбросить более 95% правил, что ускоряет и упрощает процесс сопоставления (4.1).

Рассмотрим пример со следующими правилами стилей:

p.error {color:red}

#messageDiv {height:50px}

div {margin:5px}

Первое правило будет помещено в карту классов, второе – в карту идентификаторов, а третье – в карту тегов.
Рассмотрим следующий код HTML:

an error occurred

this is a message

Сначала найдем правила для элемента p. В карте классов содержится ключ error, по которому находим правило p.error. Правила, соответствующие элементу div, содержатся в карте идентификаторов (по ключу id) и в карте тегов. Осталось только определить, какие из правил, найденных по ключам, являются подходящими.
Предположим, правило для элемента div таково:

table div {margin:5px}

Мы в любом случае извлекли бы его из карты тегов, так как ключом является крайний правый селектор, однако оно не подошло бы для этого элемента div, потому что для него не существует родительской таблицы.

Такая оптимизация используется и в WebKit, и в Firefox.

Применение правил в порядке приоритета

Свойства объекта style отвечают всем визуальным атрибутам (всем атрибутам CSS, но на более универсальном уровне). Если свойство не определяется ни одним из подходящих правил, в некоторых случаях оно может быть унаследовано от родительского объекта style. В других случаях используется значение по умолчанию.

Сложности начинаются, если существует более одного определения, и тогда, чтобы разрешить конфликт, требуется установить порядок приоритета.

Порядок приоритета таблиц стилей

Объявление свойства объекта style может содержаться сразу в нескольких таблицах стилей, иногда по нескольку раз в одной таблице. В таком случае очень важно установить верный порядок применения правил. Такой порядок называется каскадным. В спецификации CSS2 указан следующий порядок приоритета (по возрастанию).

1. Объявления браузера
2. Обычные объявления пользователя
3. Обычные объявления автора
4. Важные объявления автора

5. Важные объявления пользователя

Объявления браузера имеют самый низкий приоритет, а объявления пользователя важнее объявлений автора, только если имеют пометку !important. Объявления с одинаковым приоритетом сортируются по степени специфичности, а затем по порядку, в котором были определены. Визуальные атрибуты HTML переводятся в соответствующие объявления CSS и обрабатываются как правила автора с низким приоритетом.

Специфичность

Специфичность селектора определена в спецификации CSS2 описанным ниже образом.

- Если объявление содержится в атрибуте style, а не в правиле с селектором, выбирается значение 1, в противном случае – 0 (= a).
- Количество атрибутов ID внутри селектора (= b).
- Количество других атрибутов и псевдоклассов внутри селектора (= c).
- Количество названий элементов и псевдоэлементов внутри селектора (= d).

Объединение этих значений в последовательность a-b-c-d (в системе счисления с большим основанием) и определяет специфичность.

Основание системы счисления определяется самым большим числом в любой из категорий. Например, если a=14, можно использовать шестнадцатеричную систему. Если a=17 (что маловероятно), потребуется система счисления по основанию 17. Такая ситуация может возникнуть, если имеется селектор такого типа: html body div div p... Но вряд ли внутри селектора будет 17 тегов.

Ниже приведено несколько примеров.

```
*             {} /* a=0 b=0 c=0 d=0 -> specificity = 0,0,0,0 */
li            {} /* a=0 b=0 c=0 d=1 -> specificity = 0,0,0,1 */
li:first-line {} /* a=0 b=0 c=0 d=2 -> specificity = 0,0,0,2 */
ul li         {} /* a=0 b=0 c=0 d=2 -> specificity = 0,0,0,2 */
ul ol+li      {} /* a=0 b=0 c=0 d=3 -> specificity = 0,0,0,3 */
h1 + *[rel=up]{} /* a=0 b=0 c=1 d=1 -> specificity = 0,0,1,1 */
ul ol li.red  {} /* a=0 b=0 c=1 d=3 -> specificity = 0,0,1,3 */
li.red.level  {} /* a=0 b=0 c=2 d=1 -> specificity = 0,0,2,1 */
#x34y         {} /* a=0 b=1 c=0 d=0 -> specificity = 0,1,0,0 */
style=""          /* a=1 b=0 c=0 d=0 -> specificity = 1,0,0,0 */
```

Сортировка правил

После сопоставления правил они сортируются согласно приоритету. В WebKit для коротких списков используется сортировка простыми обменами, а для длинных – сортировка слиянием. При сортировке WebKit переопределяет для правил оператор >:

```
static bool operator >(CSSRuleData& r1, CSSRuleData& r2)
{
```

```
    int spec1 = r1.selector()->specificity();
    int spec2 = r2.selector()->specificity();
    return (spec1 == spec2) : r1.position() > r2.position() : spec1 > spec2;
}
```

Многоэтапное применение правил

В WebKit используется специальный флаг, который указывает, загружены ли все таблицы стилей верхнего уровня (включая @imports). Если совмещение уже началось, а таблица стилей еще не загружена целиком, используются заполнители, а в документе появляются соответствующие пометки. После завершения загрузки таблицы заполнители пересчитываются.

Компоновка

Когда только что созданный объект отображения включается в дерево, он не имеет ни размера, ни положения. Расчет этих значений называется компоновкой (layout или reflow).

В HTML используется поточная модель компоновки, то есть в большинстве случае геометрические данные можно рассчитать за один проход. Элементы, встречающиеся в потоке позднее, не влияют на геометрию уже обработанных элементов, поэтому компоновку можно выполнять слева направо и сверху вниз. Существуют исключения: например, для компоновки таблиц HTML может потребоваться более одного цикла (3.5).

Система координат рассчитывается на основе корневого фрейма. Используются верхняя и левая координаты.

Компоновка выполняется в несколько циклов. Она начинается с корневого объекта отображения, соответствующего элементу <html> в HTML-документе. Затем обрабатывается иерархия фреймов (или отдельные ее части), и геометрическая информация рассчитывается для объектов отображения, которым она необходима.

Корневой объект отображения имеет координаты (0; 0), а его размеры соответствуют области просмотра (видимой части окна браузера).

Любой объект отображения может при необходимости вызвать метод layout или reflow для своих дочерних элементов.

Система "грязных битов"

Чтобы не выполнять перекомпоновку при каждом изменении, браузеры используют так называемую систему "грязных битов". Измененный объект отображения и его дочерние элементы помечаются как "грязные", то есть требующие перекомпоновки.

Используется два флага: dirty и children are dirty. Флаг children are dirty означает, что перекомпоновка требуется не самому объекту отображения, а одному или нескольким из его дочерних объектов.

Глобальная и инкрементная компоновка

Если компоновка выполняется для всего дерева отображения, она называется глобальной. Ее могут вызывать перечисленные ниже события.

1. Глобальное изменение стиля, который используется во всех объектах отображения, например изменение шрифта.

2. Изменение размеров экрана.

При инкрементной компоновке изменяются только "грязные" объекты отображения (при этом может потребоваться перекомпоновка некоторых других объектов).
Инкрементная компоновка выполняется асинхронно и начинается при обнаружении "грязных" объектов отображения. Пример: после получения содержания из сети и его добавления в дерево DOM в дереве отображения появляется новый объект.

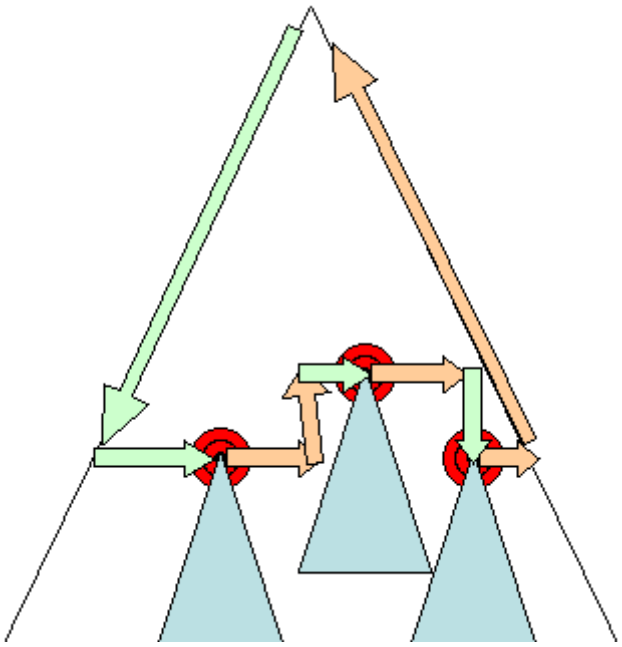


Рисунок . Инкрементная компоновка, при которой обрабатываются только "грязные" объекты отображения и их дочерние элементы (3.6).

Синхронная и асинхронная компоновка

Инкрементная компоновка выполняется асинхронно. В Firefox команды инкрементной компоновки помещаются в очередь, а затем планировщик вызывает их все вместе. В WebKit выполнение инкрементной компоновки также откладывается, чтобы обработать целое дерево за один цикл и перекомпоновать все "грязные" объекты отображения.
Скрипты, запрашивающие данные о стилях, такие как offsetHeight, могут привести к синхронному выполнению инкрементной компоновки.
Глобальная компоновка обычно выполняется синхронно.
Иногда компоновка выполняется в обратном вызове после исходной компоновки, потому что меняются значения некоторых атрибутов, таких как положение прокрутки.

Оптимизация

Если компоновка вызвана событием resize или изменением положения (но не размера) объекта отображения, размеры объекта извлекаются из кэша и не рассчитываются заново.
Если меняется только часть дерева, перекомпоновка всего дерева не выполняется. Это происходит, если изменение носит локальный характер и не влияет на окружающие объекты, например при вводе текста в текстовые поля (в остальных случаях ввод каждого символа вызывает перекомпоновку всего дерева).

Процесс компоновки

Компоновка обычно выполняется по описанной ниже схеме.

1. Родительский объект отображения определяет собственную ширину.

2. Родительский объект отображения обрабатывает дочерние элементы:
    1. определяет положение дочернего объекта отображения (задает его координаты x и y);
    2. вызывает компоновку дочернего элемента (если он помечен как "грязный", если выполняется глобальная перекомпоновка и т. д.), в результате чего рассчитывается его высота.
3. На основе суммарной высоты дочерних элементов, а также высоты полей и отступов рассчитывается высота родительского объекта отображения: она требуется его собственному родительскому объекту.
4. Биты больше не помечаются как "грязные".

В Firefox в качестве параметра компоновки используется объект nsHTMLReflowState. Помимо прочих значений, он определяет ширину родительского элемента.
В результате компоновки в Firefox создается объект nsHTMLReflowMetrics, содержащий значение высоты объекта отображения.

Расчет ширины

Ширина объекта отображения рассчитывается на основе ширины контейнера, свойства width объекта отображения, размеров полей и рамок.
Рассмотрим, как вычисляется ширина следующего элемента div:

В WebKit она будет рассчитана так (метод calcWidth класса RenderBox).

- Ширина контейнера представляет собой большее из значений availableWidth и 0. В данном случае значение свойства availableWidth равно значению contentWidth, которое рассчитывается следующим образом:

clientWidth() - paddingLeft() - paddingRight()

Значения свойств clientWidth и clientHeight соответствуют внутренним размерам объекта, исключая рамку и полосу прокрутки.

- Ширина элементов определяется атрибутом width объекта style. Ее абсолютное значение рассчитывается на основе процентной доли от ширины контейнера.
- Добавляются горизонтальные рамки и отступы.

До этого момента мы занимались расчетом предпочтительной ширины. Теперь рассчитаем ее минимальное и максимальное значение.
Если предпочтительная ширина превышает максимальную, то используется значение максимальной, а если она меньше минимальной (самого маленького неделимого объекта) – значение минимальной ширины.

Эти данные хранятся в кэше на случай, если потребуется перекомпоновка без изменения ширины.

Перенос строк

Если в процессе компоновки объект отображения обнаруживает, что необходим перенос строки, компоновка останавливается, а родительскому элементу передается запрос на перенос строки. Родительский элемент создает дополнительные объекты отображения и выполняет их компоновку.

Отрисовка

На этапе отрисовки для каждого объекта отображения по очереди вызывается метод paint и их содержание выводится на экран. Для отрисовки используется компонент инфраструктуры пользовательского интерфейса.

Глобальная и инкрементная отрисовка

При глобальной отрисовке все дерево отрисовывается целиком, а при инкрементной – только отдельные объекты отображения, не влияющие на остальные части дерева. Измененный объект отображения помечает свой прямоугольник как недействительный. Операционная система расценивает его как "грязную" область и вызывает событие paint. Области при этом объединяются, чтобы отрисовку можно было выполнить сразу для всех. В браузере Chrome отрисовка выполняется несколько сложнее, так как объект отображения находится вне главного процесса: Chrome в некоторой степени имитирует поведение операционной системы. Компонент визуального представления прослушивает эти события и делегирует сообщение корневому объекту отображения. Все объекты дерева по очереди проверяются, пока не будет найден нужный. Затем выполняется отрисовка его самого и, как правило, его дочерних элементов.

Порядок отрисовки

Порядок отрисовки определен в спецификации CSS2. Фактически он соответствует порядку помещения элементов в контексты стеков. Порядок отрисовки играет важную роль, так как стеки отрисовываются задом наперед. Порядок добавления блочных объектов в стек таков:

1. Цвет фона
2. Фоновое изображение
3. Рамка
4. Дочерние объекты
5. Внешние границы

Список отображения Firefox

В Firefox на основе анализа дерева отображения создается список отображения для отрисовываемого прямоугольника. В нем содержатся объекты отображения этого прямоугольника, расположенные в нужном порядке (сначала фон, потом рамки и т. д.). Благодаря этому для повторной отрисовки фона, фоновых изображений, рамок и т. д. достаточно пройти дерево все один раз.

В Firefox процесс оптимизирован за счет того, что элементы, которые будут скрыты (например, под непрозрачными элементами), не добавляются.

Хранилище прямоугольников в WebKit

Перед повторной отрисовкой старый прямоугольник сохраняется в WebKit как растровое изображение, а затем отрисовываются только различия между старым и новым прямоугольником.

Динамические изменения

При наступлении изменений браузеры стараются не выполнять лишних операций. Например, при изменении цвета одного элемента остальные не отрисовываются заново. При изменении

положения элемента выполняется повторная компоновка и отрисовка его самого, его дочерних элементов и, возможно, других объектов того же уровня. При добавлении узла DOM выполняется его повторная компоновка и отрисовка. Серьезные изменения, такие как увеличение размера шрифта элемента html, ведут к очистке кэша и повторной компоновке и отрисовке целого дерева.

Потоки модуля отображения

Модуль отображения работает с одним потоком: в нем выполняется почти все, кроме сетевых операций. В Firefox и Safari это основной поток браузера, в Chrome – основной процесс вкладки. Сетевые операции могут выполняться в нескольких параллельных потоках. Количество параллельных соединений ограничено и обычно составляет от 2 до 6 (например, в Firefox 3 их используется 6).

Цикл событий

Основной поток браузера представляет собой цикл событий – бесконечный цикл, который поддерживает рабочие процессы. Он ожидает отправки событий (таких как layout и paint), чтобы их обработать. Так выглядит код Firefox для основного цикла событий:

```
while (!mExiting)
  NS_ProcessNextEvent(thread);
```

Визуальная модель CSS2

Холст

Согласно спецификации CSS2, под холстом (canvas) подразумевается пространство, где отображается отформатированная структура, то есть область, в которой браузер отрисовывает содержание. Сам по себе холст бесконечен, однако браузеры обычно определяют для него ширину исходя из размеров области просмотра.

Согласно приложению к спецификации, если холст находится внутри другого холста, то он прозрачен, а в остальных случаях окрашен в определенный браузером цвет.

Модель окна в CSS

Модель окна в CSS описывает прямоугольные окна, которые создаются для элементов, содержащихся в дереве документа, и компонуются согласно модели визуального форматирования.
В каждом окне есть область для содержания (текста, картинок и т. д.) и место для необязательных отступов, рамок и полей.

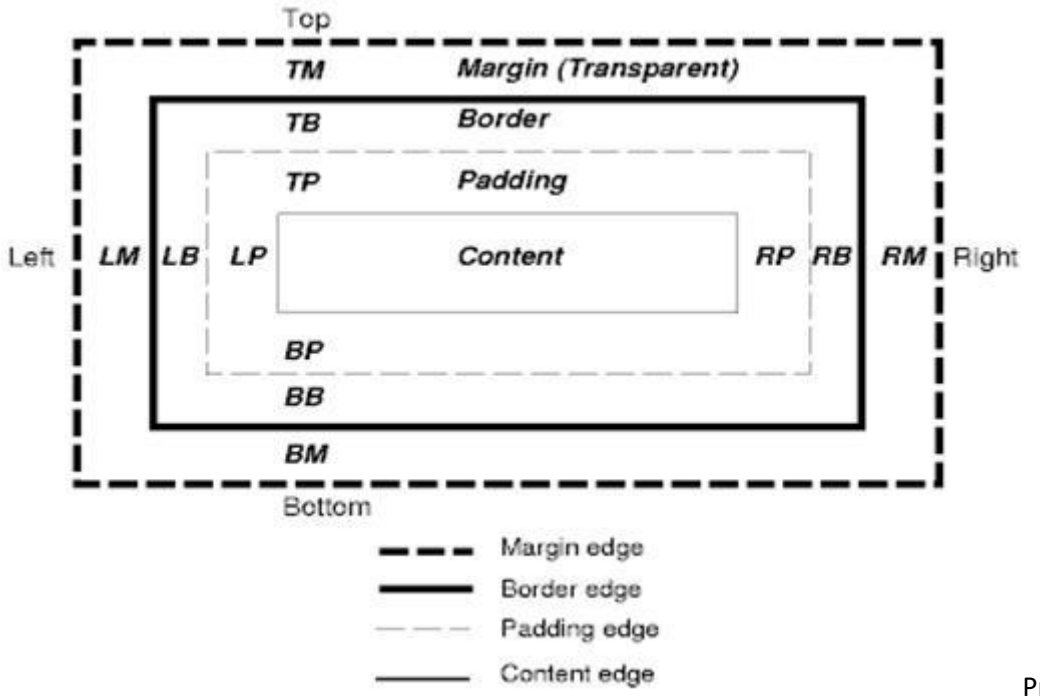


Рисунок . Модель окна в CSS2.

В каждом узле создается от 0 до n таких окон.
У всех элементов есть свойство display, определяющее тип окна, которое необходимо создать.
Примеры:

block - generates a block box.

inline - generates one or more inline boxes.

none - no box is generated.

По умолчанию окна являются строчными элементами, однако в таблицах стилей браузера могут быть заданы иные значения по умолчанию. Например, элемент div по умолчанию является блочным.
Пример таблицы стилей со значениями по умолчанию можно найти здесь: www.w3.org/TR/CSS2/sample.html.

Схема позиционирования

Существует три схемы позиционирования.

1. Стандартная: объект размещается в документе согласно своему положению в дереве отображения и в дереве DOM, а также своему типу и размерам окна.
2. Плавающая: объект сначала компонуется по стандартной схеме, затем смещается в крайнее правое или крайнее левое положение.
3. Абсолютная: положение объекта в дереве отображения отличается от его положения в дереве DOM.

Схема позиционирования определяется свойством position и атрибутом float.

- Значения static и relative соответствуют стандартной схеме.
- Значения absolute и fixed соответствуют абсолютной схеме.

При выборе значения static положение не задается: используется значение по умолчанию. В остальных схемах автор может указать положение с помощью значений top, bottom, left и right.

Способ компоновки окна определяется следующими факторами:

- Тип окна
- Размеры окна
- Схема позиционирования
- Внешняя информация (размеры изображения, размер экрана)

Типы окон

Блочное окно создает собственный блок прямо в окне браузера.

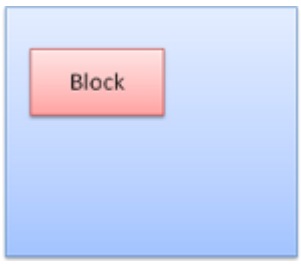


Рисунок . Блочное окно.

Строчное окно не имеет собственного блока и помещается внутрь контейнера.

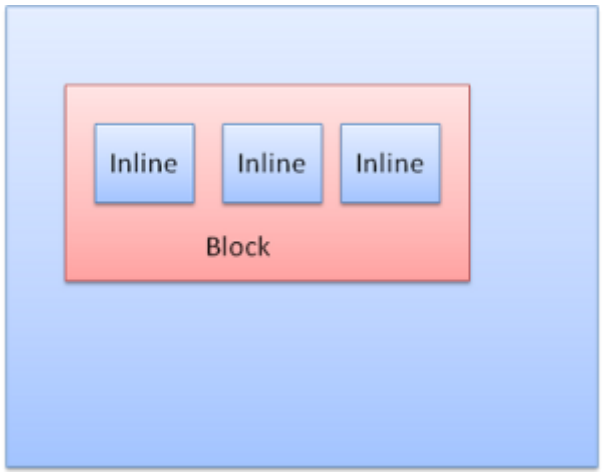


Рисунок . Строчные окна.

При форматировании блочные окна размещаются друг под другом, а строчные – друг рядом с другом.

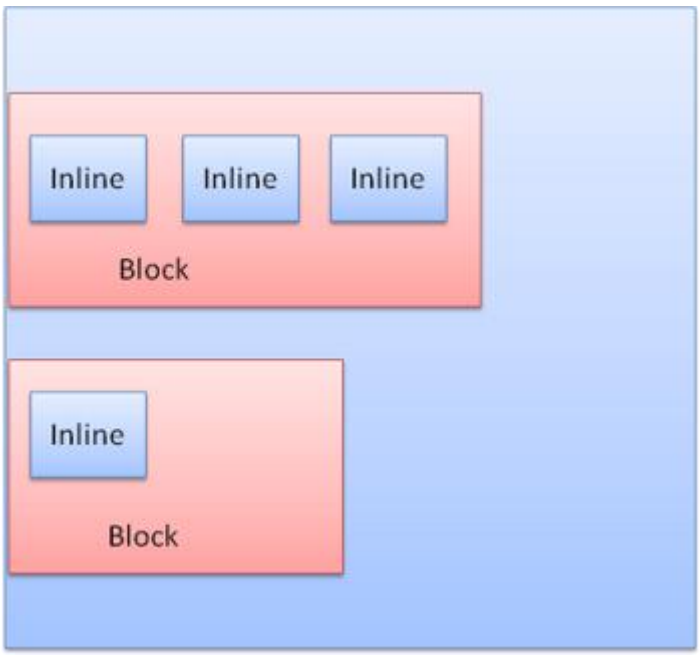


Рисунок . Форматирование блочных и строчных окон.

Строчные окна объединяются в строки. Высота строки должна быть больше или равна высоте самого высокого окна, когда окна выровнены по нижнему краю. Если ширина контейнера недостаточна, чтобы вместить все строчные окна, они переносятся на следующие строки. Типичным примером является абзац.

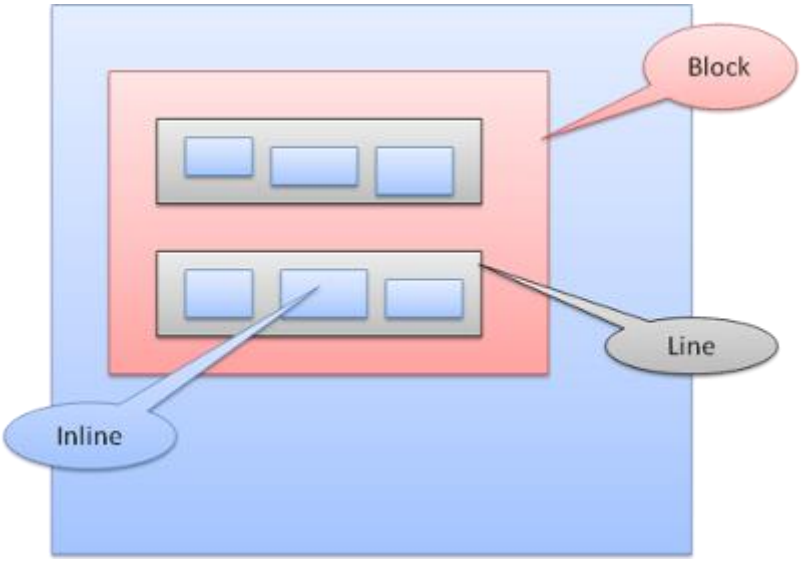


Рисунок . Строки.

Позиционирование

Относительное позиционирование

Относительное позиционирование означает, что объект размещается стандартным способом, а затем смещается на нужное расстояние.

```
<html>
        <div>
                <span>1</span>
                <span>2</span>
                <span style="position:relative;left:5px">3</span>
        </div>
</html>
```

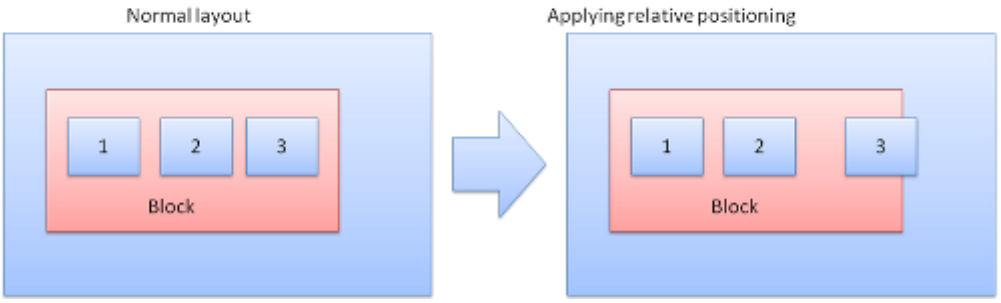


Рисунок . Относительное позиционирование.

Плавающие элементы

Плавающее окно смещается вправо или влево в пределах строки. Что интересно, остальное содержание обтекает его. Рассмотрим код HTML:

```
<p>
 <img style="float:right" src="images/image.gif" width="100" height="100">
 Lorem ipsum dolor sit amet, consectetuer...
</p>
```

На веб-странице он будет выглядеть так:

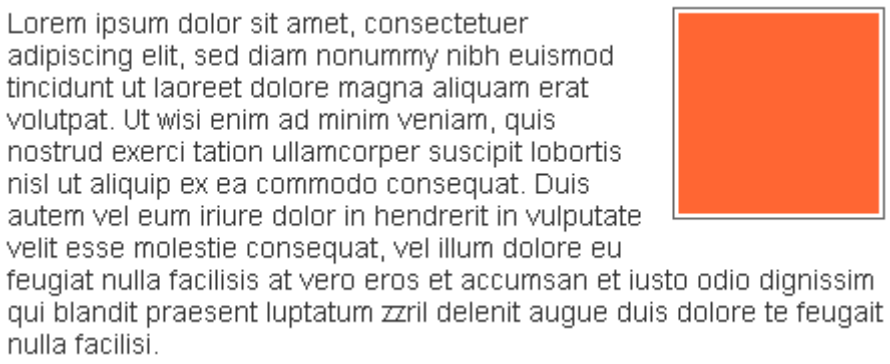


Рисунок . Плавающие элементы.

Абсолютное и фиксированное позиционирование

Компоновка элемента определена вне зависимости от стандартной схемы: элемент просто исключается из нее. Его размеры зависят от размеров контейнера. В случае фиксированного позиционирования контейнером служит область просмотра.

```
<html>
        <div>
                <span>1</span>
                <span>2</span>
                <span style="position:fixed;top:5px,left:5px">3</span>
        </div>
</html>
```

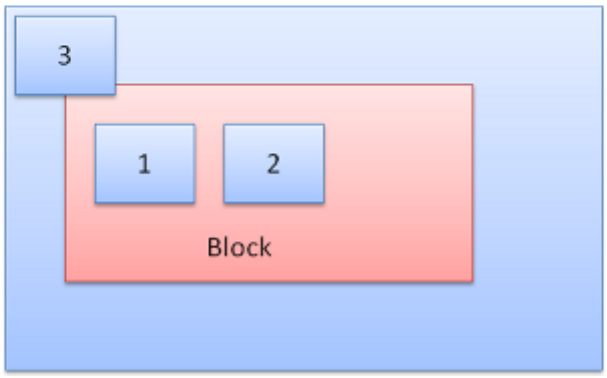


Рисунок .

Фиксированное позиционирование.
Обратите внимание: фиксированное окно остается на месте даже при прокрутке документа!

Многослойное представление

Эта возможность реализуется свойством z-index в CSS. Оно соответствует третьему измерению, или оси z.

Окна делятся на стеки (стековые контексты). В каждом стеке сначала отрисовываются элементы на заднем плане, а затем на переднем (расположенные ближе к пользователю). Если они перекрываются, элементы на заднем плане не будут видны.
Порядок стеков определяется свойством z-index. Окна со свойством z-index формируют локальный стек, а область просмотра представляет собой внешний стек.

Пример:

```
<style type="text/css">
    div {
      position: absolute;
      left: 2in;
      top: 2in;
    }
</style>

<p>
  <div
      style="z-index: 3;background-color:red; width: 1in; height: 1in; ">
  </div>
```

```
  <div
    style="z-index: 1;background-color:green;width: 2in; height: 2in;">
  </div>
</p>
```

На веб-странице это будет выглядеть так:

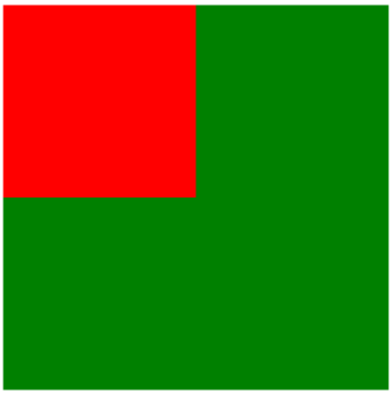


Рисунок . Фиксированное позиционирование.

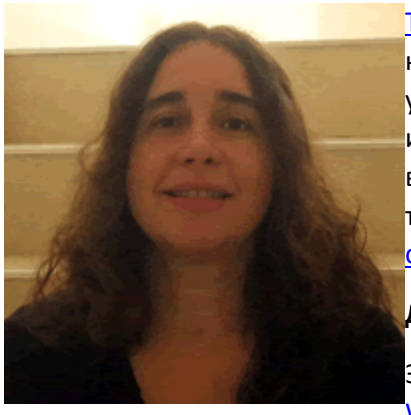
Красный элемент div определен в коде раньше зеленого и в стандартной схеме был бы отрисован первым, однако значение его свойства z-index выше, поэтому он находится в стеке, который ближе к пользователю.

Ресурсы

1. Архитектура браузеров
    1. Grosskurth, Alan. A Reference Architecture for Web Browsers (pdf)
    2. Gupta, Vineet. How Browsers Work - Part 1 - Architecture
2. Синтаксический анализ
    1. Alfred Aho, Ravi Sethi, Jeffrey Ullman. Compilers: Principles, Techniques, and Tools ("Dragon book"), Addison-Wesley, 1986
    2. Rick Jelliffe. The Bold and the Beautiful: two new drafts for HTML 5.
3. Firefox
    1. L. David Baron, Faster HTML and CSS: Layout Engine Internals for Web Developers.
    2. L. David Baron, Faster HTML and CSS: Layout Engine Internals for Web Developers (видеозапись технической презентации в Google)
    3. L. David Baron, Mozilla's Layout Engine
    4. L. David Baron, Mozilla Style System Documentation
    5. Chris Waterson, Notes on HTML Reflow
    6. Chris Waterson, Gecko Overview
    7. Alexander Larsson, The life of an HTML HTTP request
4. WebKit

1. David Hyatt, Implementing CSS(part 1)
2. David Hyatt, An Overview of WebCore
3. David Hyatt, WebCore Rendering
4. David Hyatt, The FOUC Problem

5. Спецификации W3C
    1. Спецификации HTML 4.01
    2. Спецификации HTML5
    3. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification
6. Инструкции по сборке браузеров
    1. Firefox. https://developer.mozilla.org/en/Build_Documentation
    2. WebKit. http://webkit.org/building/build.html



Тали Гарсиэль работает программистом в Израиле. Она начала карьеру в 2000 году, когда познакомилась с "неудобной" уровневой моделью Netscape. Как и Ричард Фейнман, она живо интересуется, что и как работает, поэтому начала изучать внутреннее устройство браузеров и записывать результаты. Тали также опубликовала краткое руководство по обработке кода на стороне клиента.

Другие языки

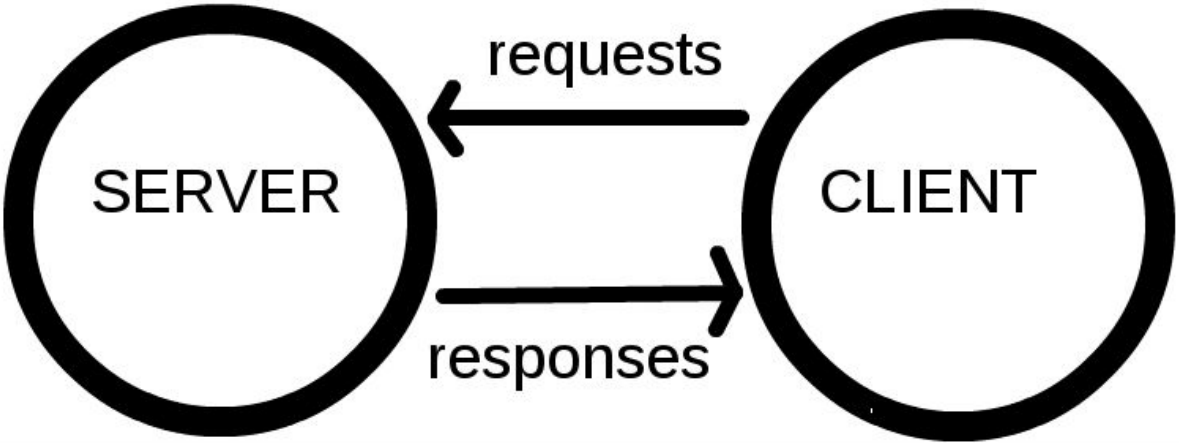
Эта страница переведена на японский. Дважды! How Browsers Work - Behind the Scenes of Modern Web Browsers (на японском), автор @_kosei_ и ブラウザってどうやって動いてるの？（モダンWEBブラウザシーンの裏側, авторы @ikeike443 и @kiyoto01. Спасибо всем!

Как работает Веб даст упрощенное представление о том, что происходит при просмотре веб-страницы в браузере на вашем компьютере или телефоне.

Эта теория не так важна для написания веб-кода в краткосрочной перспективе, но в скором времени вы действительно начнете извлекать выгоду из понимания того, что происходит в фоновом режиме.

Клиенты и серверыРаздел

Компьютеры, подключенные к сети называются клиентами и серверами. Упрощенная схема того, как они взаимодействуют, может выглядеть следующим образом:



- Клиенты являются обычными пользователями, подключенными к Интернету посредством устройств (например, компьютер подключен к Wi-Fi, или ваш телефон подключен к мобильной сети) и программного обеспечения, доступного на этих устройствах (как правило, браузер, например, Firefox или Chrome).
- Серверы - это компьютеры, которые хранят веб-страницы, сайты или приложения. Когда клиентское устройство пытается получить доступ к веб-странице, копия страницы загружается с сервера на клиентский компьютер для отображения в браузере пользователя.

Остальные части панели инструментовРаздел

Клиент и сервер, о которых мы рассказали выше, не расскрывают всю суть. Есть много других компонентов, и мы опишем их ниже.

А сейчас давайте представим, что Веб - это дорога. Одна сторона дороги является клиентом, который представляет собой ваш дом. Другая сторона дороги является сервером, который представляет собой магазин. Вы хотите что-то купить в нём.

Помимо клиента и сервера, мы также должны уделить внимание:

- **Ваше Интернет-подключение**: Позволяет отправлять и принимать данные по сети. Оно подобно улице между домом и магазином.
- **TCP/IP**: Протокол Управления Передачей и Интернет Протокол являются коммуникационными протоколами, которые определяют, каким образом данные должны передаваться по сети. Они как транспортные средства, которые позволяют сделать заказ, пойти в магазин и купить ваши товары. В нашем примере, это как автомобиль или велосипед (или собственные ноги).
- **DNS**: Система Доменных Имён напоминает записную книжку для веб-сайтов. Когда вы вводите веб-адрес в своем браузере, браузер обращается к DNS, чтобы найти реальный адрес веб-сайта, прежде чем он сможет его получить. Браузеру необходимо выяснить, на каком сервере живет сайт, поэтому он может отправлять HTTP-сообщения в нужное место (см. Ниже). Это похоже на поиск адреса магазина, чтобы вы могли попасть в него.
- **HTTP**: Протокол Передачи Гипертекста - это протокол, который определяет язык для клиентов и серверов, чтобы общаться друг с другом. Он, как язык, который вы используете, чтобы заказать ваш товар.
- **Файлы компонентов**: сайт состоит из нескольких различных файлов, которые подобны различным отделам с товарами в магазине. Эти файлы бывают двух основных типов:
    - **Файлы кода**: сайты построены преимущественно на HTML, CSS и JavaScript, хотя вы познакомитесь с другими технологиями чуть позже.
    - **Материалы**: это собирательное название для всех других вещей, составляющих сайт, такие как изображения, музыка, видео, документы Word и PDF.

Что же на самом деле происходит?Раздел

Когда вы вводите веб-адрес в свой браузер (для нашей аналогии - посещаете магазин):

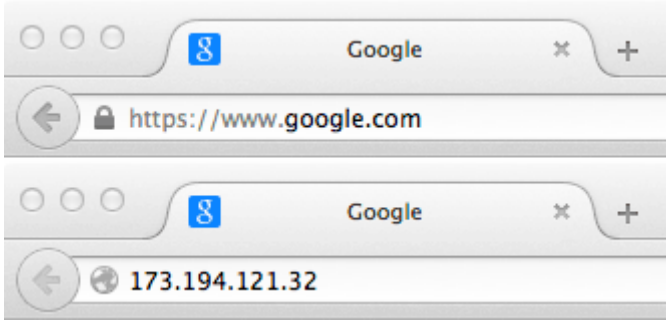
1. Браузер обращается к DNS серверу и находит реальный адрес сервера, на котором "живет" сайт (Вы находите адрес магазина).
2. Браузер посылает HTTP запрос к серверу, запрашивая его отправить копию сайта для клиента (Вы идёте в магазин и заказываете товар). Это сообщение и все остальные данные, передаваемые между клиентом и сервером, передаются по интернет-соединению с использованием протокола TCP/IP.
3. Если запрос клиента корректен, сервер отправляет клиенту статус "200 OK", который означает: "Конечно, вы можете посмотреть на этот сайт! Вот он", а затем начинает отправку файлов сайта в браузер в виде небольших порций, называемых пакетными данными (магазин выдает вам ваш товар или вам привозят его домой).
4. Браузер собирает маленькие куски в полноценный сайт и показывает его вам (товар прибывает к вашей двери — новые вещи, потрясающе!).

DNSРаздел

Реальные веб-адреса - неудобные, незапоминающиеся строки, которые Вы вводите в адресную строку, чтобы найти ваши любимые веб-сайты. Эти строки состоят из чисел, например: 63.245.215.20.

Такой набор чисел называется IP-адресом и представляет собой уникальное местоположение в Интернете. Впрочем, его не очень легко запомнить, правда? Вот почему изобрели DNS. Это специальные сервера, которые связывают веб-адрес, который вы вводите в браузере (например, "mozilla.org"), с реальным IP-адресом сайта.

Сайты можно найти непосредственно через их IP-адреса. Попытайтесь зайти на сайт Mozilla, набрав 63.245.215.20 в адресной строке на новой вкладке браузера.



ПакетыРаздел

Ранее мы использовали термин "пакеты", чтобы описать формат, в котором данные передаются от сервера к клиенту. Что мы имеем в виду? В основном, когда данные передаются через Интернет, они отправляются в виде тысячи мелких кусочков, так что множество разных пользователей могут скачивать один и тот же сайт одовременно. Если бы сайты отправлялись одним большим куском, тогда бы только один пользователь мог скачать его за один раз, и это, очевидно, сделало бы пользование интернетом не эффективным и не очень радостным.