Как устроен современный браузер?

Слайд 1

Сейчас браузерами пользуются, наверное, все пользователи Интернета. И естественно ни один человек задавался вопросом, а как же это все работает? И вот на рассвете таких браузеров, как Firefox, Safari, Chrome, да и вообще всего веба, масштабное исследование провела израильская веб-программистка Тали Гарсиэль.

Слайд 2

Она начала карьеру в 2000 году, когда познакомилась с "неудобной" уровневой моделью Netscape. 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.

И сейчас я попробую более-менее кратко передать все ее мысли и результаты исследования.

Начнем с вопроса, а для чего же нам все-таки нужен браузер?

Слайд 3

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. 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.

Слайд 4

Теперь рассмотрим основные компоненты браузера.

Слайд 5

(*Рисунок 1. Основные компоненты браузера.*)

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.
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 HTML5 имеется определение термина "веб-база данных": это полноценная (хотя и облегченная) браузерная база данных.

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

А теперь о каждом поговорим подробнее.

Слайд 6

Пользовательские интерфейсы разных браузеров имеют много общего. К интерфейсу относятся все элементы, кроме окна, в котором отображается запрашиваемая страница.

Основные элементы интерфейса браузера перечислены на слайде и обозначены на рисунке.

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.

Слайд 7

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.

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.

Слайд 8

Теперь об основной схеме работы модуля отображения.

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

Схема дальнейшей работы модуля отображения выглядит приведенным на экране образом. (*Рисунок 2. 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.

Слайд 9

(*Рисунок 3. Схема работы модуля отображения WebKit. Рисунок 4. Схема работы модуля отображения Mozilla Gecko.*)

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

Небольшое отличие Gecko состоит в том, что между HTML-файлом и деревом DOM находится еще один уровень. Он называется буфером содержания и служит для формирования элементов DOM.

Теперь поговорим об этапах работы подробнее.

Слайд 10

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.

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 синтаксическим деревом.

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 бесконтекстную грамматику. Естественные языки не подчиняются правилам бесконтекстной грамматики, поэтому стандартные техники синтаксического анализа для них не годятся. Для этого существуют анализаторы.

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.

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.

Слайд 11

Генераторы

Существуют специальные приложения для создания синтаксических анализаторов, так как сделать это вручную не так-то просто. Они называются генераторами. Достаточно загрузить в генератор грамматику языка, и он автоматически создаст анализатор.

В WebKit используется два известных генератора: Flex для создания лексического и Bison для создания синтаксического анализатора. Во Flex загружается файл с определениями токенов, а в Bison – синтаксические правила языка в формате BNF.

Но ни один из стандартных анализаторов не подходит для языка HTML. Его невозможно определить с помощью бесконтекстной грамматики, с которой работают стандартные синтаксические анализаторы. Поэтому для определения HTML существует формальный стандарт – формат DTD (Document Type Definition), грамматика которого как раз является контекстной.

DOM

Полученное в результате синтаксического анализа 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, который представлен на экране: (*текст*).

Дерево DOM для этой разметки представлено ниже. (*Рисунок 6. Дерево DOM для разметки из примера.*)

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.

Слайд 12

На странице HTML вы никогда не увидите ошибку "Недопустимый синтаксис", так как браузеры умеют корректировать ошибки содержания, не прерывая работу. На экране перечислены основные типы ошибок, которые определены в спецификации HTML5. Большая часть кода синтаксического анализатора служит как раз 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.

Слайд 13

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.

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 таблицы стилей, содержащие правила CSS. Объект правил CSS содержит селектор и объявление, а также другие объекты, характерные для грамматики CSS.

(*Рисунок 7. Синтаксический анализ CSS.*)

Слайд 14

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 является механизмом оптимизации. 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, а лишь обрабатывает ссылки на внешние ресурсы, такие как внешние скрипты, таблицы стилей и картинки.

Так как таблицы стилей 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.

Слайд 15

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.

В Firefox элемент дерева отображения называется "фреймом". В WebKit используется термин "объект отображения". Каждый объект отображения располагает данными об отрисовке самого себя и своих дочерних элементов.

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. Объекты отображения, указывающие на объекты style, содержат негеометрическую информацию. Учитывается и тип элемента: например, для элементов управления формами и таблиц используются специальные фреймы.

Объекты обработки соответствуют элементам DOM, но не идентичны им. Невизуальные элементы DOM не включаются в дерево отображения. Кроме того, в дерево не включаются элементы, у которых для свойства display задан атрибут none.

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.

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.

Viewport (область просмотра) – это главный контейнер. В WebKit он представлен объектом RenderView.

(*Рисунок 8. Дерево отображения и соответствующее ему дерево DOM.*)

Слайд 16

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

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

Как браузеры справляются с этой задачей?

Слайд 17

Правила стилей извлекаются из нескольких источников, перечисленных на экране.

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, 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.
Такая классификация упрощает поиск подходящих правил. Нам не приходится проверять все объявления: достаточно извлечь из карты подходящие правила.

Свойства объекта style отвечают всем визуальным атрибутам. 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.

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

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.

Слайд 18

Когда только что созданный объект отображения включается в дерево, он не имеет ни размера, ни положения. Расчет этих значений называется **компоновкой.**

В HTML используется поточная модель компоновки, то есть в большинстве случаев геометрические данные можно рассчитать за один проход. 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. Его размеры соответствуют видимой части окна браузера. Затем обрабатывается иерархия фреймов (или отдельные ее части), и геометрическая

информация рассчитывается для объектов отображения, которым она необходима.

Чтобы не выполнять перекомпоновку при каждом изменении, браузеры помечают измененный объект отображения и его дочерние элементы как "грязные", то есть требующие перекомпоновки.

Если компоновка выполняется для всего дерева отображения, она называется глобальной. Ее могут вызывать: глобальное изменение стиля, который используется во всех объектах отображения, например изменение шрифта. Или 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.

Команды инкрементной компоновки помещаются в очередь, а затем планировщик вызывает их все вместе. Глобальная компоновка обычно выполняется синхронно. Также скрипты, запрашивающие данные о стилях, могут привести к синхронному выполнению инкрементной компоновки.

(*Рисунок 10. Инкрементная компоновка, при которой обрабатываются только "грязные" объекты отображения и их дочерние элементы.*)

Слайд 19

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.

При глобальной отрисовке все дерево отрисовывается целиком, а при инкрементной – только отдельные объекты отображения, не влияющие на остальные части дерева. Также как при компоновке, 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 объектов и их дочерних элементов.

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

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.

Основной поток браузера представляет собой цикл событий – бесконечный цикл, который поддерживает рабочие процессы. Он ожидает отправки событий (таких как компоновка и отрисовка), чтобы их обработать.

Слайд 20

Визуальная модель CSS2

Слайд 21

Согласно спецификации CSS2, под **холстом** 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.

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.

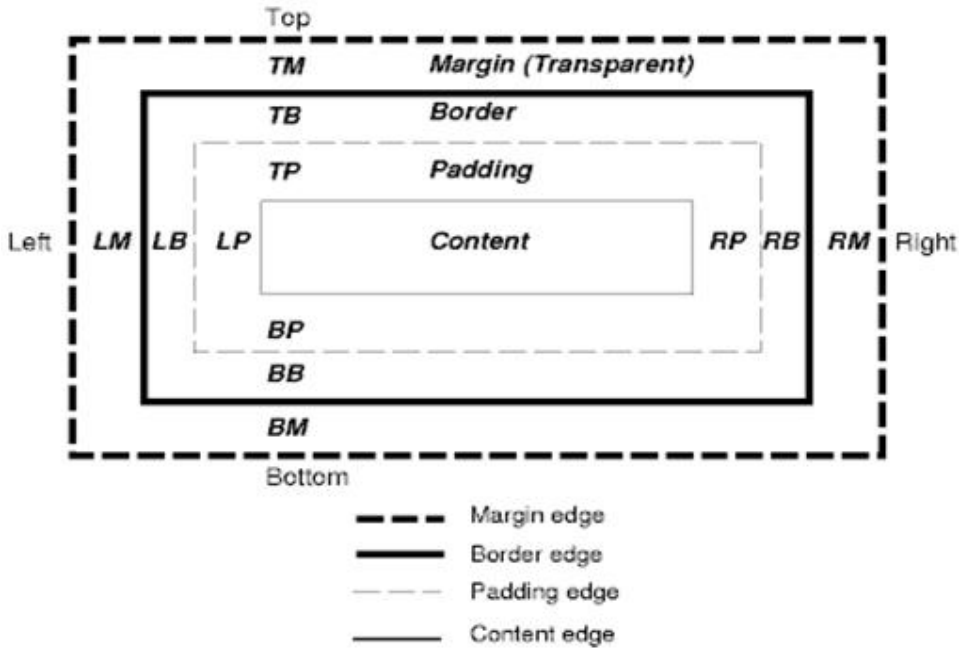

Рисунок 11. Модель окна в CSS2.

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

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.

Слайд 22

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.

Слайд 23

Блочное окно создает собственный блок прямо в окне браузера. (*Рисунок 12. Блочное окно.*) Строчное окно не имеет собственного блока и помещается внутрь контейнера. (*Рисунок 13. Строчные окна.*) При форматировании блочные окна размещаются друг под другом, а строчные – друг рядом с другом. (*Рисунок 14. 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 15. Строки.*)

Слайд 24

Относительное **позиционирование** объекта означает, что объект размещается стандартным способом, а затем смещается на нужное расстояние. (*Рисунок 16. Относительное позиционирование.*) При абсолютном и фиксированном позиционировании 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. (*Рисунок 18. Фиксированное позиционирование.*) **Обратите внимание**: фиксированное окно остается на месте даже при прокрутке документа!

Слайд 25

Главным ресурсом, как я уже говорила, стала статья израильской программистки Тали Гарсиэль. Кто хочет узнать еще больше информации, интернет вам в помощь, так как я рассказала наверно процентов 20 от того, что было написано в статье.

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