Как устроен современный браузер? Есть ли будущее у веб-приложений?

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.

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.

Основное предназначение 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. 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.

Основные компоненты браузера

1. **Пользовательский интерфейс**

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

Основные элементы интерфейса браузера перечислены ниже и обозначены на рисунке.

1. Адресная строка для ввода URI

2. Кнопки навигации "Назад" и "Вперед"

3. Закладки

4. Кнопки обновления и остановки загрузки страницы

5. 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.

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

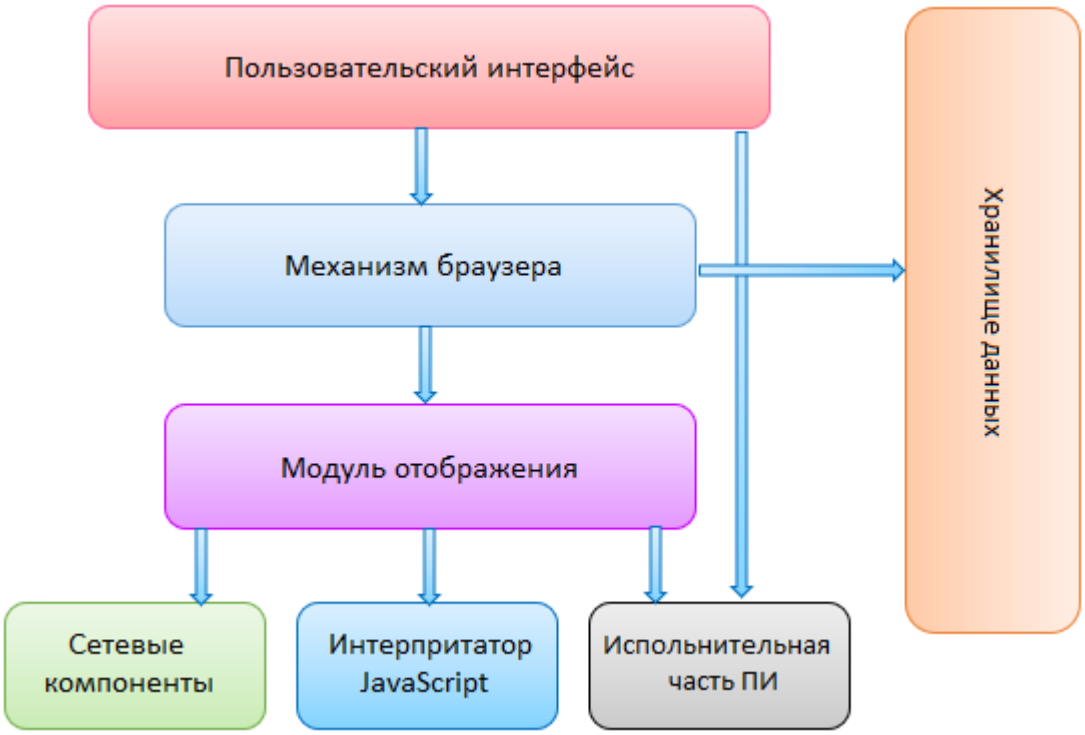


Рисунок 1. Основные компоненты браузера.

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.

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.

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.

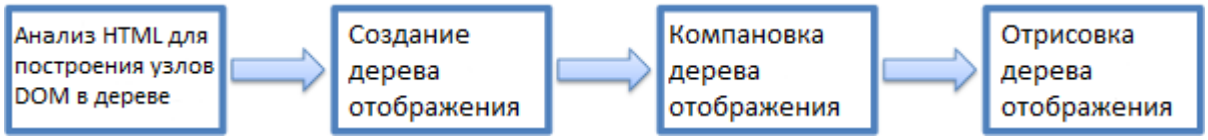


Рисунок 2. Схема работы модуля отображения.

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.

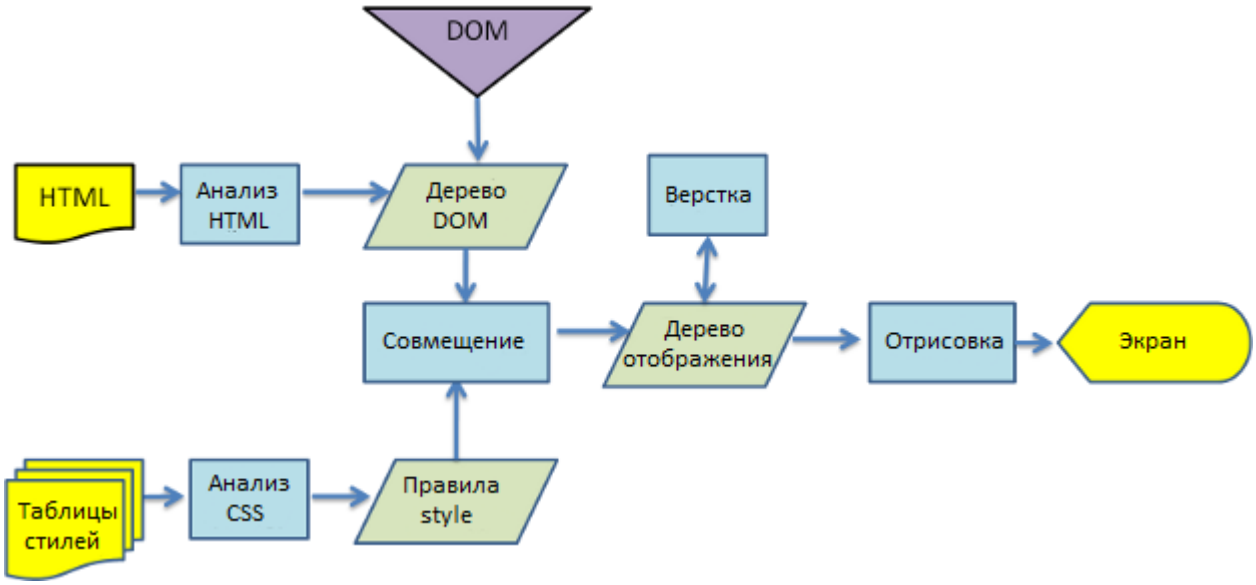


Рисунок 3. Схема работы модуля отображения WebKit.

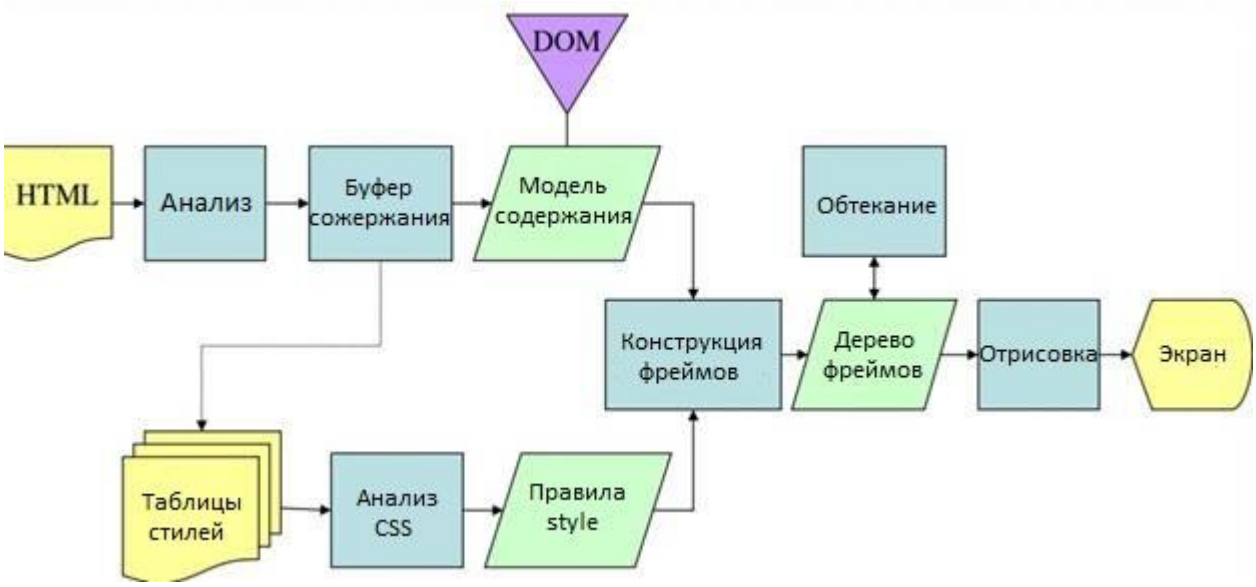


Рисунок 4. Схема работы модуля отображения Mozilla Gecko.

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

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 и служит для формирования элементов DOM.

Анализ документа

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 деревом синтаксического анализа, или просто 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.

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.

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:

```
<html>
  <body>
    <p>
```

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

Дерево DOM для этой разметки выглядит так:

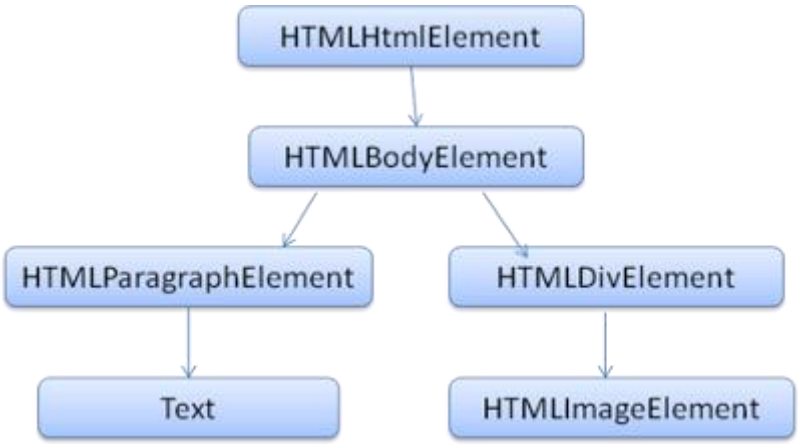


Рисунок 6. Дерево DOM для разметки из примера.

Как и в случае HTML, спецификации DOM разрабатывает Консорциум W3C. Это универсальная спецификация для работы с документами. В специальном модуле описаны элементы, характерные для 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.

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.

Так как многие документы HTML содержат ошибки, синтаксический анализатор 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.*
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.*

Синтаксический анализ CSS

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

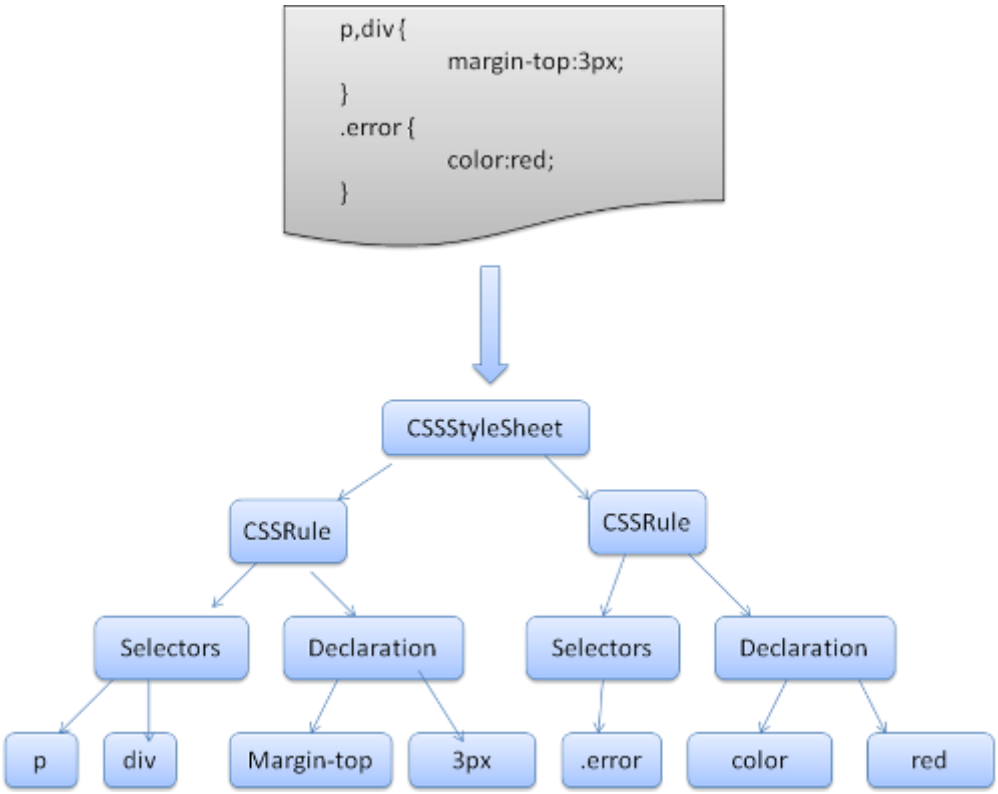


Рисунок 7. Синтаксический анализ CSS.

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.

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.

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 один 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.

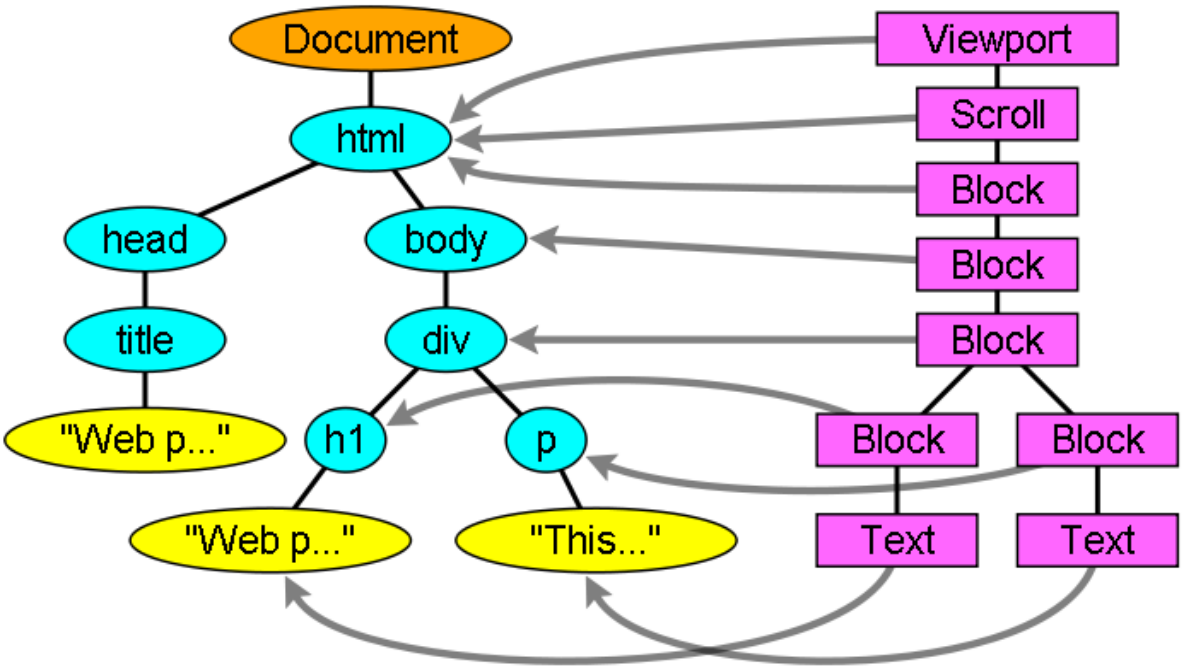


Рисунок 8. Дерево отображения и соответствующее ему дерево DOM.
Viewport (область просмотра) – это главный контейнер. В WebKit он представлен объектом RenderView.

Вычисление стилей

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?

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, 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.

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.

После сопоставления правил они сортируются согласно приоритету. В WebKit для коротких списков используется сортировка простыми обменами, а для длинных – сортировка слиянием.

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 случаев 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.

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.

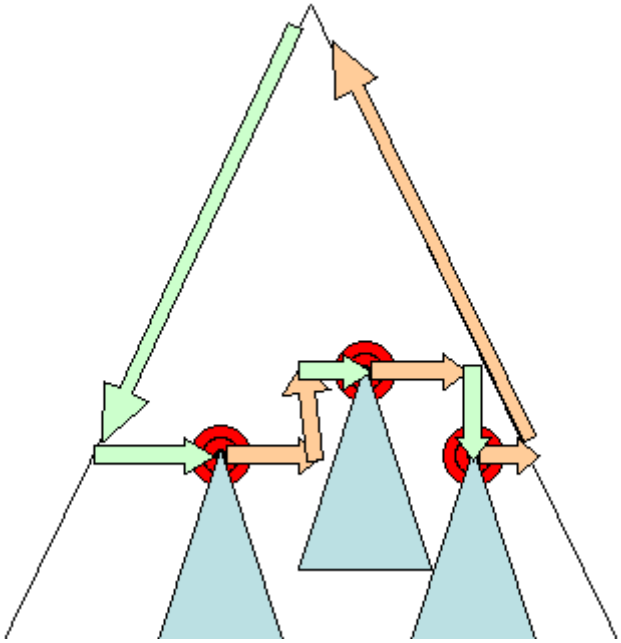


Рисунок 10. Инкрементная компоновка, при которой обрабатываются только "грязные" объекты отображения и их дочерние элементы.

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, 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".

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 всего 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.

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.

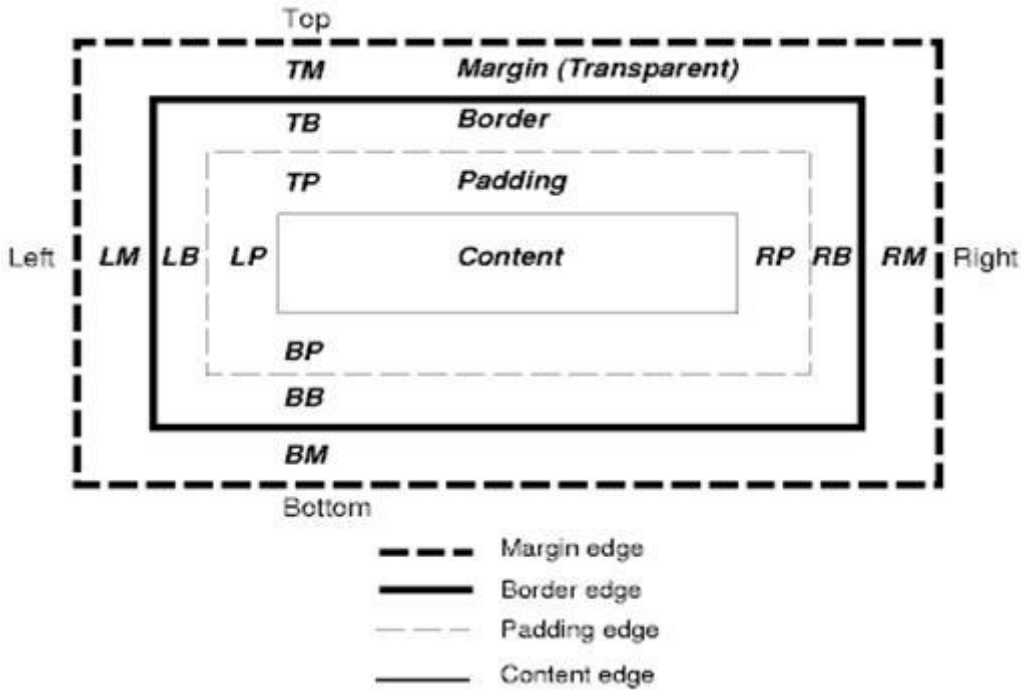


Рисунок 11. Модель окна в CSS2.

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.

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.

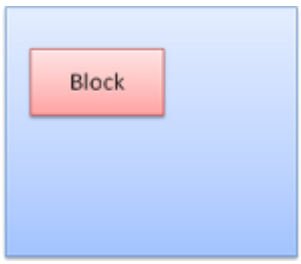


Рисунок 12. Блочное окно.

Строчное окно не имеет собственного блока и помещается внутрь контейнера.

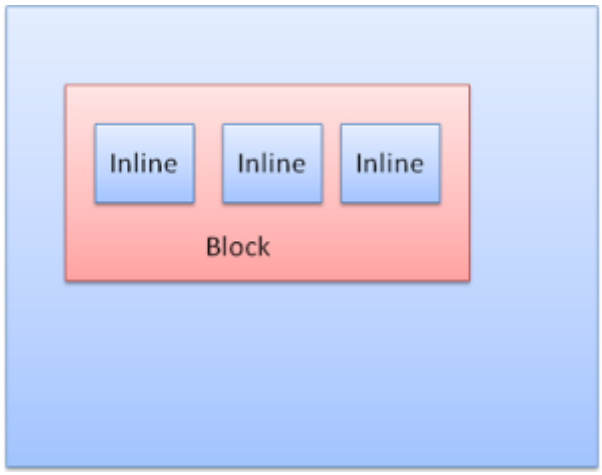


Рисунок 13. Строчные окна.

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

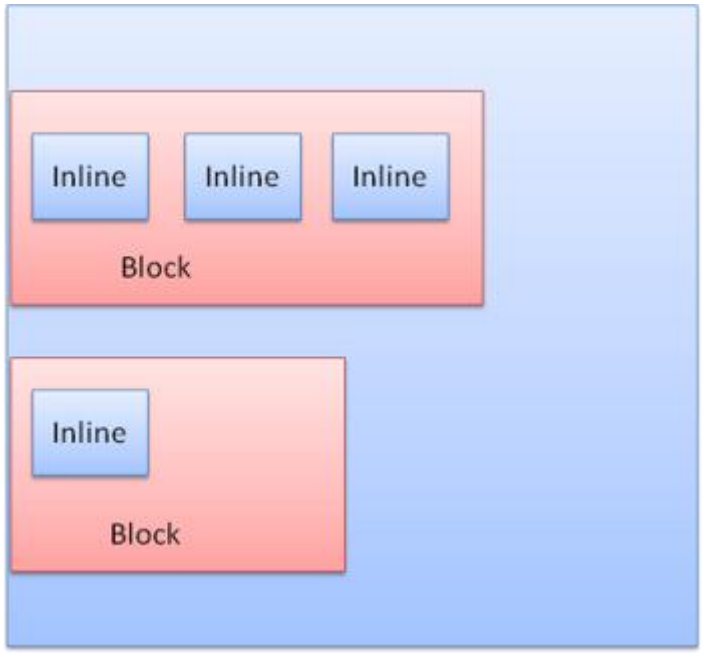


Рисунок 14. Форматирование блочных и строчных окон.

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.

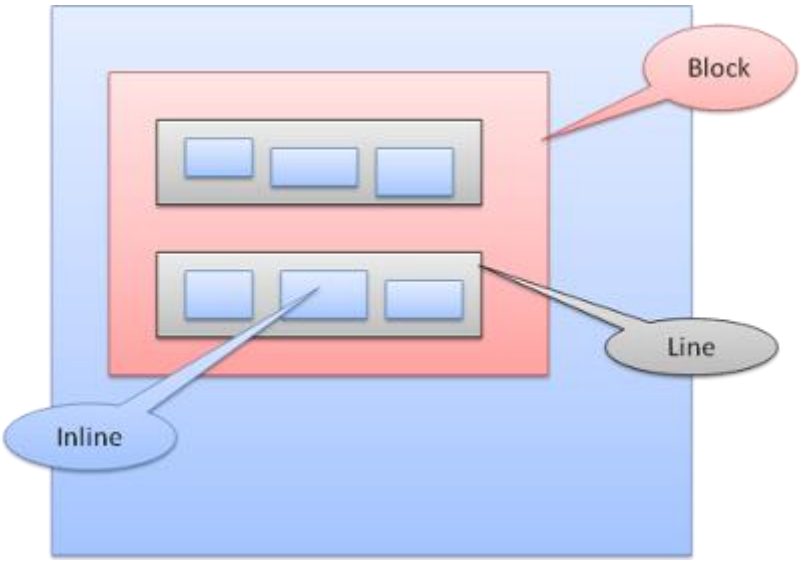


Рисунок 15. Строки.

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>
```

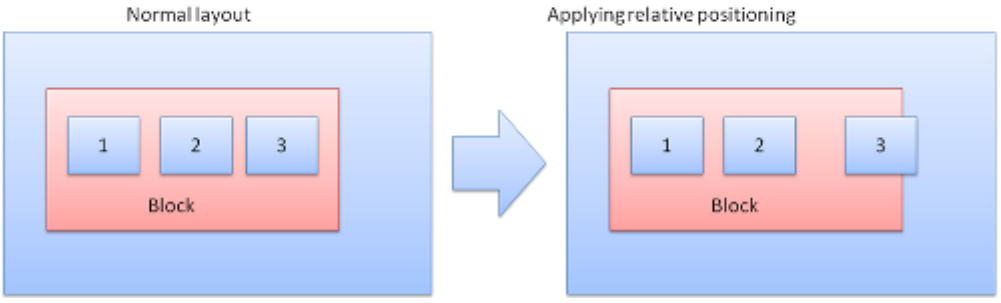


Рисунок 16. Относительное позиционирование.

Плавающие элементы

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

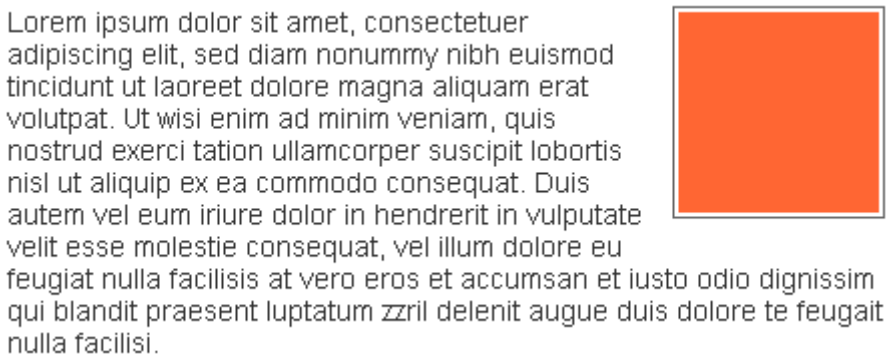


Рисунок 17. Плавающие элементы.

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>
```

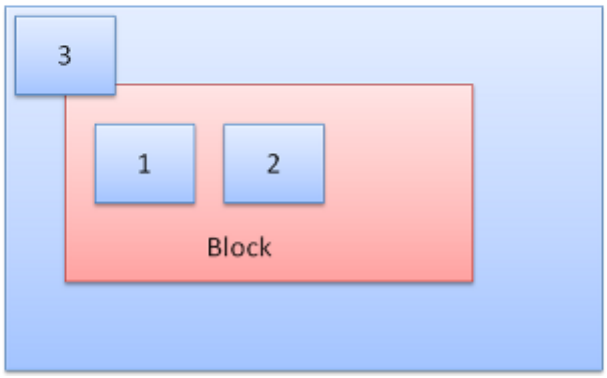


Рисунок 18. Фиксированное позиционирование.
Обратите внимание: фиксированное окно остается на месте даже при прокрутке документа!

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.

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

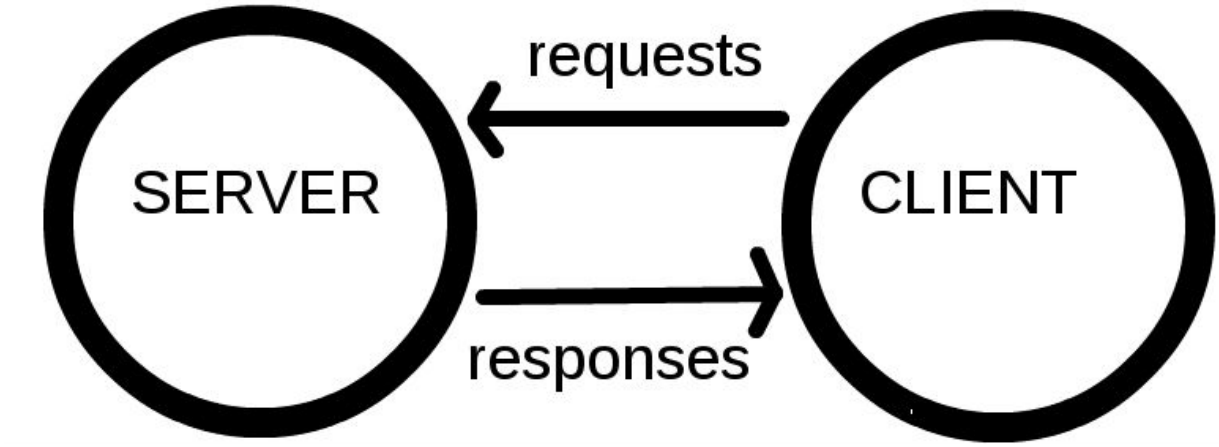
6. Инструкции по сборке браузеров
    1. Firefox. https://developer.mozilla.org/en/Build_Documentation

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 инструментов

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?

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

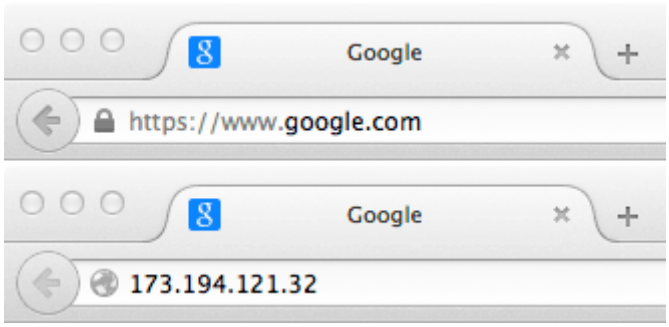
1. Браузер обращается к DNS серверу и находит реальный адрес сервера, на котором "живет" сайт (Вы находите адрес магазина).
2. Браузер посылает HTTP запрос к серверу, запрашивая его отправить копию сайта для клиента (Вы идёте в магазин и заказываете товар). Это сообщение и все остальные данные, передаваемые 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!).

DNS

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.



Пакеты

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.