1. **Принципы маршрутизации данных в Интернете. OSPF**

**Маршрутизация** — процесс определения маршрута данных в сетях связи.

Маршруты могут задаваться административно (статические маршруты), либо вычисляться с помощью алгоритмов маршрутизации, базируясь на информации о топологии и состоянии сети, полученной с помощью протоколов маршрутизации (динамические маршруты). В динамической маршрутизации учитывают расстояния до узла, состояние сети, скорость передачи канала связи, стоимость и другие параметры. Если топология сети изменится, то маршрутизаторы смогут определить это и вычислить новый маршрут.

Плюсы - автоматически определяется топология сети и вычисляется маршрут, экономит время, оперативно реагирует на изменения в сети.

Минусы - значительно расходуются ресурсы процессора.

Маршрутизация в компьютерных сетях выполняется специальными программно-аппаратными средствами - маршрутизаторами; в простых конфигурациях может выполняться и компьютерами общего назначения, соответственно настроенными.

Маршрутизатор составляет таблицу маршрутизации, где указывается следующий маршрутизатор для пересылки пакета. Пакет будет передаваться по сети от одного маршрутизатора к другому пока не достигнет того маршрутизатора, который знает, что запрашиваемый компьютер или другое сетевое устройство подключено именно к нему (к маршрутизатору). Маршрутизаторы передают пакеты только на основе адреса сети, а не адреса всего хоста. Иначе таблица маршрутизации содержала бы сотни тысяч данных о каждом сетевом узле. Поэтому в таблице маршрутизации хранится только адрес сети.Если сервер А отправит пакеты серверу В, то маршрутизатор, приняв пакет, анализирует адрес получателя и проверяет свою таблицу маршрутизации. Если в ней найдена сеть, которая соответствует адресу сети получателя, то пакет пересылается соседнему маршрутизатору. Если запись не найдена, то пакет уничтожается

Статическими маршрутами могут быть:

* маршруты, не изменяющиеся во времени;
* маршруты, изменяющиеся по расписанию;

Плюсы - позволяет снизить затраты процессора на вычисление таблицы, высокая скорость работы.

Минусы - невозможно отслеживать состояние сети и оперативно на это реагировать, кроме того, это трудоемкий процесс в больших сетях.

Для вычисления маршрутов используются такие протоколы, как RIP, OSPF, IS-IS, EIGRP, BGP. Они также работают на сетевом уровне, за исключением BGP.

**OSPF** — протокол динамической маршрутизации, основанный на технологии отслеживания состояния канала.

Протокол OSPF был разработан IETF в 1988 году. Последняя версия протокола представлена в RFC 2328 (1998 год). Протокол OSPF представляет собой протокол внутреннего шлюза (Interior Gateway Protocol). Протокол OSPF распространяет информацию о доступных маршрутах между маршрутизаторами одной автономной системы.

OSPF имеет следующие преимущества:

* Высокая скорость сходимости;
* Поддержка сетевых масок переменной длины;
* Оптимальное использование пропускной способности с построением дерева кратчайших путей.

Для понимания необходимости данных «зон» при проектировании сети, необходимо понять, как OSPF работает. Есть несколько понятий, связанных с этим протоколом, которые не встречаются в других протоколах и являются уникальными:

* **Router ID:** Уникальный 32-х битный номер, назначенный каждому маршрутизатору. Как правило, это сетевой адрес с интерфейса маршрутизатора, обладающий самым большим значением. Часто для этих целей используется loopback интерфейс маршрутизатора.
* **Маршрутизаторы-соседи:** Два маршрутизатора с каналом связи между ними, могут посылать друг другу сообщения.
* **Соседство:** Двухсторонние отношения между маршрутизаторами-соседями. Соседи не обязательно формируют между собой соседство.
* **LSA: Link State Advertisement** – сообщение о состоянии канала между маршрутизаторами.
* **Hello сообщения:** С помощью этих сообщений маршрутизаторы определяют соседей и формируют LSA
* **Area (Зона):** Некая иерархия, набор маршрутизаторов, которые обмениваются LSA с остальными в одной и той же зоне. Зоны ограничивают LSA и стимулируют агрегацию роутеров.

OSPF – протокол маршрутизации с проверкой состояния каналов. Представьте себе карту сети – для того, чтобы ее сформировать, OSPF совершает следующие действия:

1. Сперва, когда протокол только запустился на маршрутизаторе, он начинает посылать hello-пакеты для нахождения соседей и выбора **DR** (назначенный маршрутизатор). Эти пакеты включают в себя информацию о соседях и состоянии каналов. К примеру, OSPF может определить соединение типа «точка-точка», и после этого в протоколе данное соединение «поднимается», т.е. становится активным. Если же это распределенное соединение, маршрутизатор дожидается выбора DR перед тем как пометить канал активным.
2. Существует возможность изменить **Priority ID** для, что позволит быть уверенным в том, что DR-ом станет самый мощный и производительный маршрутизатор. В противном случае, победит маршрутизатор с самым большим IP-адресом. Ключевая идея **DR** и **BDR (Backup DR)**, заключается в том, что они являются единственными устройствами, генерирующими LSA и они обязаны обмениваться базами данных состояния каналов с другими маршрутизаторами в подсети. Таким образом, все не-DR маршрутизаторы формируют соседство с DR. Весь смысл подобного дизайна в поддержании масштабируемости сети. Очевидно, что единственный способ убедиться в том, что все маршрутизаторы оперируют одной и той же информацией о состоянии сети – синхронизировать БД между ними. В противном случае, если бы в сети было 35 маршрутизаторов, и требовалось бы добавить еще одно устройство, появилась бы необходимость в установлении 35 процессов соседства. Когда база централизована (т.е существует центральный, выбранный маршрутизатор - DR) данный процесс упрощается на несколько порядков.
3. **Обмен базами данных** – крайне важная часть процесса по установлению соседства, после того как маршрутизаторы обменялись hello-пакетами. При отсутствии синхронизированных баз данных могут появиться ошибки, такие как петли маршрутизации и т.д. Третья часть установления соседства – обмен LSA. Это понятие будет разобрано в следующей статье, главное, что необходимо знать – нулевая зона (Area 0) особенная, и при наличии нескольких зон, все они должны быть соединены с Area 0. Так же это называется магистральной зоной.

**Типы маршрутизаторов OSPF**

* ***ABR***(Area Border Router) – маршрутизатор внутри нулевой зоны, через который идет связь с остальными зонами
* ***DR, BDR***(Designated Router, Backup Designated Router) – этот тип маршрутизаторов обсуждался выше, это основной и резервирующий маршрутизаторы, которые ответственны за базу данных маршрутизаторов в сети. Они получают и посылают обновления через Multicast остальным маршрутизаторам в сети.
* ***ASBR***(Autonomous System Boundary Router) – этот тип маршрутизаторов соединяет одну или несколько автономных систем для осуществления возможного обмена маршрутами между ними.

1. **Прокси-сервер: использование, виды**

Прокси-сервер — промежуточный сервер (комплекс программ) в компьютерных сетях, выполняющий роль посредника между пользователем и целевым сервером, позволяющий клиентам как выполнять косвенные запросы (принимая и передавая их через прокси-сервер) к другим сетевым службам, так и получать ответы. Сначала клиент подключается к прокси-серверу и запрашивает какой-либо ресурс, расположенный на другом сервере. Затем прокси-сервер либо подключается к указанному серверу и получает ресурс у него, либо возвращает ресурс из собственного кэша (в случаях, если прокси имеет свой кэш). В некоторых случаях запрос клиента или ответ сервера может быть изменён прокси-сервером в определённых целях. Прокси-сервер позволяет защищать компьютер клиента от некоторых сетевых атак и помогает сохранять анонимность клиента, но также может использоваться мошенниками для скрытия адреса сайта, уличённого в мошенничестве, изменения содержимого целевого сайта (подмена), а также перехвата запросов самого пользователя.

Чаще всего прокси-серверы применяются для следующих целей:

* обеспечение доступа компьютеров локальной сети к сети Интернет;
* кэширование данных: если часто происходят обращения к одним и тем же внешним ресурсам для снижения нагрузки на канал во внешнюю сеть и ускорения получения клиентом запрошенной информации;
* сжатие данных: прокси-сервер загружает информацию из Интернета и передаёт информацию конечному пользователю в сжатом виде для экономии трафика;
* защита локальной сети от внешнего доступа: например, можно настроить прокси-сервер так, что локальные компьютеры будут обращаться к внешним ресурсам только через него, а внешние компьютеры не смогут обращаться к локальным вообще (они «видят» только прокси-сервер);
* ограничение доступа из локальной сети к внешней: например, можно запрещать доступ к определённым веб-сайтам, фильтровать рекламу и вирусы;
* анонимизация доступа к различным ресурсам: прокси-сервер может скрывать сведения об источнике запроса или пользователе. В таком случае целевой сервер видит лишь информацию о прокси-сервере, например IP адрес, но не имеет возможности определить истинный источник запроса;
* обход ограничений доступа: используется, например, пользователями стран, где доступ к некоторым ресурсам ограничен законодательно и фильтруется.

Прокси-сервер, к которому может получить доступ любой пользователь сети интернет, называется открытым.

***Прозрачный прокси*** — схема связи, при которой трафик или его часть перенаправляется на прокси-сервер неявно (средствами маршрутизатора). При этом клиент может использовать все преимущества прокси-сервера без дополнительных настроек браузера (или другого приложения для работы с интернетом). Пример создания маршрута для такой схемы: route -p add 10.32.5.5 mask 255.255.255.255 10.32.1.14.

***Обратный прокси*** — прокси-сервер, который, в отличие от прямого, ретранслирует запросы клиентов из внешней сети на один или несколько серверов, логически расположенных во внутренней сети. Часто используется для балансировки сетевой нагрузки между несколькими веб-серверами и повышения их безопасности, играя при этом роль межсетевого экрана на прикладном уровне.

***Веб-прокси*** — широкий класс прокси-серверов, выполненных в форме веб-приложения.

***Искажающие прокси-серверы*** - передают целевому серверу ложную информацию об истинном пользователе;

1. **Веб-сервера. Статический и Динамический контент**

Понятие «веб-сервер» может относиться как к аппаратной начинке, так и к программному обеспечению. Или даже к обеим частям, работающим совместно.

1. С точки зрения "железа", «веб-сервер» — это компьютер, который хранит файлы сайта и доставляет их на устройство конечного пользователя (веб-браузер и т.д.). Он подключен к сети Интернет и может быть доступен через доменное имя.
2. С точки зрения ПО, *веб-сервер* включает в себя несколько компонентов, которые контролируют доступ веб-пользователей к размещенным на сервере файлам, как минимум — это *HTTP-сервер*. *HTTP-сервер* — это часть ПО, которая понимает веб-адреса и HTTP.

На самом базовом уровне, когда браузеру нужен файл, размещенный на веб-сервере, браузер запрашивает его через HTTP-протокол. Когда запрос достигает нужного веб-сервера ("железо"), сервер HTTP (ПО) принимает запрос, находит запрашиваемый документ и отправляет обратно, также через HTTP. Чтобы опубликовать веб-сайт, необходим либо статический, либо динамический веб-сервер.

**Статический веб-сервер**, или стек, состоит из компьютера ("железо") с сервером HTTP (ПО). Мы называем это «статикой», потому что сервер посылает размещенные файлы в браузер «как есть».

**Динамический веб-сервер** состоит из статического веб-сервера и дополнительного программного обеспечения, чаще всего *сервера приложения* и *базы данных*. Мы называем его «динамическим», потому что сервер приложений изменяет исходные файлы перед отправкой в ваш браузер по HTTP.

Cервер может отдавать статическое или динамическое содержимое. **«Статическое»** означает «отдается как есть». **Статический сайт** — сайт, состоящий из статичных html (htm, dhtml, xhtml) страниц, составляющих единое целое. Содержит в себе (в виде HTML-размеченных) текст, изображения, мультимедиа содержимое (аудио, видео) и HTML-теги. Теги бывают как служебные, предназначенные для обозревателя, так и предназначенные для размещения, формирования внешнего вида и отображения информации. Все изменения на сайт вносятся в исходный код документов (страниц) сайта, для чего необходимо иметь доступ к файлам на веб сервере.

**«Динамическое»** означает, что сервер обрабатывает данные или даже генерирует их на лету из базы данных. Это обеспечивает большую гибкость, но технически сложнее в реализации и обслуживании. **Динамический сайт** — состоит из динамичных страниц - контента, скриптов и прочего, в большинстве случаев в виде отдельных файлов. Страница сайта, показываемая в итоге браузеру пользователя, формируется на стороне сервера динамически, по запросу, из страницы-шаблона и отдельно хранимого содержимого (информации, скриптов и др.). Как правило, для отображения любого количества однотипных страниц используется одна страница-шаблон, в которую подгружается соответствующее содержимое.

**Процесс получения содержимого сайта обычно выглядит следующим образом:**

1. Генерация содержимого на стороне сервера;
2. Передача сгенерированной странички клиенту;
3. Генерация содержимого на стороне клиента.

### Генерация содержимого на стороне сервера

Сервер получает запрос от Клиента (например, ***page.ru/index.php***) и запускает обработку файла-скрипта (в данном случае — ***index.php***) интерпретатором. Языки программирования на Серверной стороне используются разные, наиболее часто встречаются, например: Java, PHP, Perl и другие.

Именно на этой стадии происходит **выборка** необходимой информации из баз данных и **наполнение** ею страницы, после чего готовая страница передаётся Клиенту.

### Генерация содержимого на стороне клиента

После того, как страница получена Клиентом с Сервера, программа-браузер обрабатывает её и отображает Пользователю, при этом *исполняя скрипты* Клиентской стороны, если они были указаны в странице и получены. На Клиентской стороне используется JavaScript

### Комбинированная генерация

Чаще всего в жизни встречается именно комбинация этих двух методов генерации — весь «новый интернет» основан на нём, это, и «умная строка с подсказкой вариантов» у поисковых систем, и всплывающие меню, и многое другое.

1. **Сеть доставки содержимого (CDN)**

Сеть доставки (и дистрибуции) содержимого (англ. Content Delivery Network или Content Distribution Network, CDN) — группа серверов, установленных в разных местах для предоставления веб-контента в широком географическом регионе. Использование контент-провайдерами CDN способствует увеличению скорости загрузки интернет-пользователями аудио-, видео-, программного, игрового и других видов цифрового содержимого в точках присутствия сети CDN.

При использовании сети CDN, данные центрального сервера интернет-ресурса реплицируются на периферийные платформы. Каждая платформа поддерживает в актуальном состоянии полную или частичную копию распространяемых данных. Узел сети, входящий в состав платформы, взаимодействует с локальными сетями интернет-провайдеров и распространяет контент конечным пользователям по кратчайшему сетевому маршруту с оптимального по загруженности сервера. Длина сетевого маршрута зависит от географической или топологической удалённости пользовательского компьютера от сервера или стоимости передачи трафика в регионе присутствия.

Кэширование является самым распространённым методом реализации CDN решения, так как предполагает оптимальное использование дискового пространства и связующих каналов сети. При этом максимальные затраты по времени загрузки файла (очередь файлов) берет на себя первый пользователь, обратившийся на оригинальный сервер контент-провайдера. Все последующие пользователи будут обращаться к уже загруженным репликам (HTTP-объектам) с ближайшего к ним сервера. Таким образом, на удалённых серверах хранится только популярный и часто запрашиваемый контент.

Крупные CDN могут состоять из огромного количества распределённых узлов и размещать свои сервера непосредственно в сети каждого локального интернет-провайдера. Многие CDN операторы делают акцент на пропускной способности связующих каналов и минимальном количестве точек присоединения в регионе присутствия. Вне зависимости от используемой архитектуры, главным предназначением подобных сетей является ускорение передачи как статического контента, так и непрерывного потока данных.

1. **Современные веб-браузеры. Принципы работы**

Браузер — это клиентская программа, позволяющая в простой форме посылать запросы серверам на загрузку веб-страниц. В задачи браузера помимо простейших операций по связи с серверами входит: обработка полученной HTML-разметки, интерпретация стилей и скриптов, контроль ошибок и по возможности их исправление, хранение пользовательской информации. Браузеры, представленные различными компаниями, могут по-разному реализовывать эти механизмы или игнорировать какие-либо из них. Такие возможности, объединенные в виде программы, называется браузерным движком.

Принцип работы браузера.

Этапы рабочего процесса браузера:

* При вводе имени сайта в адресной строке, клике по ссылке в поисковой системе или на любом сайте, браузер посылает запрос серверу на загрузку определенной страницы
* Сервер получает запрос и проверяет, есть ли такая страница
* Сервер осуществляет передачу HNML-разметки страницы браузеру
* Браузер обрабатывает разметку и выводит результат пользователю
* Механизм рендеринга

Ниже перечислены основные компоненты браузера.

1. **Пользовательский интерфейс**
2. **Механизм браузера** – управляет взаимодействием интерфейса и модуля отображения.
3. **Модуль отображения** – отвечает за вывод запрошенного содержания на экран. Модуль отображения получает содержание запрошенного документа по протоколу сетевого уровня, обычно фрагментами по 8 КБ. 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. Cинтаксический анализ является важным этапом работы модуля. 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.

**Таже есть синтаксические анализаторы HTML и CSS**. Задача синтаксического анализатора HTML – переводить информацию из кода HTML в синтаксическое дерево. Так же работает анализатор CSS

1. **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.
2. **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.
3. **Интерпретатор JavaScript** – используется для синтаксического анализа и выполнения кода JavaScript.
4. **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.

### Построение дерева отображения(после анализа синтаксиса)

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.

**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.

**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.

### 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).

1. **Хранение данных на клиенте. Cookie. Недостатки Cookie**

Ку́ки — небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя. Веб-клиент (обычно веб-браузер) всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в составе HTTP-запроса. В техническом плане cookie представляют собой фрагменты данных, изначально отправляемых веб-сервером браузеру. При каждом последующем посещении сайта браузер пересылает их обратно серверу. Без cookie каждый просмотр веб-страницы является изолированным действием, не связанным с просмотром других страниц того же сайта, с помощью же cookie можно выявить связь между просмотром разных страниц. Кроме отправки cookie веб-сервером, cookie могут создаваться скриптами, если они поддерживаются и включены в браузере. Применяется для сохранения данных на стороне пользователя, на практике обычно используется для:

* аутентификации пользователя;
* хранения персональных предпочтений и настроек пользователя;
* отслеживания состояния сеанса доступа пользователя;
* сведения статистики о пользователях.

Поддержки браузерами cookie (приём, сохранение и последующая пересылка серверу сохранённых cookie) требуют многие сайты с ограничениями доступа, большинство интернет-магазинов. Настройка оформления и поведения многих веб-сайтов по индивидуальным предпочтениям пользователя тоже основана на cookie.

МИНУСЫ:

1. Cookie легко перехватить и подменить (например, для получения доступа к учётной записи), если пользователь использует нешифрованное соединение с сервером. В группе риска пользователи, выходящие в интернет при помощи публичных точек доступа Wi-Fi и не использующие при этом таких механизмов, как SSL и TLS. Шифрование позволяет также решить и другие проблемы, связанные с безопасностью передаваемых данных.
2. Неточная идентификация - если на компьютере используется более одного браузера, то, как правило, каждый имеет отдельное хранилище для cookie. Поэтому cookie идентифицируют не человека, а сочетание учётной записи, компьютера, и браузера. Таким образом, любой человек, который использует несколько учётных записей, компьютеров или браузеров, имеет несколько наборов cookie.
3. Межсайтовые cookie - Каждый сайт должен иметь свои собственные cookie, и сайт example1.com не должен изменять или устанавливать cookie другого сайта example2.org. Уязвимости веб-браузеров позволяют вредоносным сайтам нарушать это правило. Это похоже на подмену cookie, но здесь злоумышленник атакует пользователей с уязвимыми браузерами, а не сайт напрямую. Целью таких атак могут быть идентификаторы сессий.
4. Нестабильность между клиентом и сервером - Cookie могут вызвать противоречия между клиентом и сервером.
5. Срок действия cookie - Постоянные cookie критикуются экспертами за свой долгий срок хранения, который позволяет веб-сайтам отслеживать пользователей и создавать их профиль с течением времени[40]. Здесь затрагиваются и вопросы безопасности, поскольку украденные постоянные cookie могут использоваться на протяжении значительного периода времени.

Большинство современных браузеров позволяет пользователям выбрать — принимать cookie или нет, но их отключение делает невозможной работу с некоторыми сайтами. Кроме того, по законам некоторых стран (например, по закону Евросоюза от 2016 года, см. общий регламент по защите данных) сайты должны в обязательном порядке запрашивать согласие перед установкой cookie.

1. **Хранение данных на клиенте. Технология Web Storage. Преимущества веб-хранилищ**

Вообще, Web Storage (известное также как DOM-хранилище) относится ко множеству API, которые нацелены на обеспечение простого способа хранения данных со стороны клиента в браузере. Эта технология более безопасная и быстрая, чем cookie, не о говоря уже о ее мощности. **Это** программные методы и протоколы веб-приложения, используемые для хранения данных в веб-браузере. Интернет-хранилище представляет собой постоянное хранилище данных, похожее на куки, но со значительно расширенной ёмкостью и без хранения информации в заголовке запроса HTTP.

* Web Storage предоставляет два отличных объекта для хранения данных:
* localStorage: Используя этот объект, вы будете сохранять данные без даты окончания срока. Это значит, что данные будут сохранены на локальном диске пользователя навсегда.

sessionStorage: Используя этот объект, данные, которые вы сохранили, будут находиться в хранилище до тех пор, пока посетитель не закроет браузер (но не вкладку). Хороший вариант использования для данного объекта это сохранение временных данных, таких как содержимое форм, заполненных пользователем, в случае, если пользователь случайно закрыл вкладку или обновил страницу.

ПРЕИМУЩЕСТВА:

* Данные сохраняются в пользовательском браузере и не передаются по сети как файлы cookie.
* Интернет-хранилище поддерживает гораздо больше места на диске в сравнении с куки, которому доступно всего 4 Кбайта, что примерно в 1000 раз меньше чем у веб-хранилища (5 Мбайт на домен в Mozilla Firefox, Google Chrome, и Opera, и 10 Мбайт в Internet Explorer[4]).
* Интернет-хранилище в настоящее время предоставляет программный интерфейс лучше, чем куки. Интерфейс представляет собой ассоциативный массив модели данных, где ключи и значения являются строками.
* Интернет-хранилище предлагает две различных области: Локальное хранилище и Сессионное хранилище, которые отличаются по своим масштабам и времени жизни. Данные размещаются в отдельное для каждого домена локальное хранилище (оно доступно для всех скриптов из домена, который первоначально добавил данные) и сохраняются после закрытия браузера. Сессия сохраняется по принципу одна страница-одно окно и ограничивается жизнью данного окна, то есть для каждого открытого окна создаётся новая сессия, которая прекращает своё существование при закрытии окна и не зависит от домена открывшего её. Сохранение сессии предназначено для предоставления отдельных экземпляров одного и того же веб-приложения для работы в разных окнах, не мешая друг другу. В случае с куки подобное становится крайне затруднительно или даже невозможно.

1. **Особенности хранения данных на стороне клиента. LocalStorage и SessionStorage. Сохранение, извлечение и удаление данных**

Отличие между этими хранилищами сводится только к периоду времени, в течение которого они могут хранить данные, помещенные в них:

SessionStorage – выполняет это в течение определённого промежутка времени (сессии). Закрытие вкладки или браузера приводит их к удалению. При этом данные в SessionStorage сохраняются при обновлении страницы.

LocalStorage – осуществляет это в течение неограниченного времени. Они сохраняются при перезагрузке браузера и компьютера. Их длительность хранения ничем не ограничена. Но, хоть эти данные могут храниться бесконечно в браузере, обычный пользователь может их очень просто удалить, например выполнив очистку истории (при включенной опции «файлы cookie и другие данные сайтов»).

Главное отличие различных типов хранилищ – время хранения данных и их доступность.

* sessionStorage хранит данные в рамках одной сессии (посещения, т.е. до закрытия пользователем окна браузера).
* localStorage позволяет хранить данные и после прекращения сеанса.

С точки зрения программирования различие в использовании сеансового и локального типов хранилищ сводится к различию имен объектов, посредством которых осуществляется доступ к ним: sessionStorage и localStorage соответственно.

**Сохранение и извлечение данных**

Для этого необходимо проверить, поддерживает ли клиент Local storage:

function isLocalStorageAvailable() {

try {

return 'localStorage' in window && window['localStorage'] !== null;

} catch (e) {

return false;

}

}

Существует несколько способов сохранения и извлечения данных:

* localStorage.setItem('id', 'value');
* localStorage['id'] = 'value';

Функция setItem принимает строку ключа и строку значения в качестве аргументов. Несмотря на то что формат данных Web Storage поддерживает передачу значений, не являющихся строковыми, в текущих версиях браузеров набор поддерживаемых типов значений, как правило, ограничен строками.

Соответственно и извлечь данные можно следующим образом:

* var data = localStorage.getItem('id');
* var data = localStorage['data'];

**Осталось только рассмотреть способы удаления данных из хранилища.**

Удаление определенного хранимого элемента:

**localStorage.removeItem('id');**

Удаление всех элементов:

* localStorage.clear();

В случае использования хранилища сеанса используются те же функции и методы.

Верстка и программирование

1. **Комплексные селекторы. Примеры**

Сложный селектор - это последовательность одного или нескольких составных селекторов, разделенных комбинаторами. Он представляет собой набор одновременных условий для набора элементов в определенных отношениях, описанных его комбинаторами. (Сложные селекторы представлены в виде <complex-selector> в грамматике селекторов.) Говорят, что данный элемент соответствует сложному селектору, когда существует список элементов, каждый из которых соответствует соответствующему составному селектору в сложном селекторе, с каждой парой элементы, последовательные в списке, соответствующие комбинатору между их соответствующими составными селекторами, причем последний элемент является данным элементом.

Вы можете использовать сложные селекторы, что означает, что вы можете исключить несколько свойств. Например, теперь можно выбрать любой элемент, который не имеет одновременно классов «.important» и «.dialog»:

:not(.important.dialog) { }

:matches(h1+h2, h2+h3, h3+h4, h4+h5) {}

.syntax-highlighted :matches(.css-keyword, .css-tag){}

1. **Псевдоклассы и псевдоэлементы CSS. Примеры**

Обычно стили CSS применяются к тем элементам веб-страницы, которые видны в ее исходном коде. Но существуют случаи, когда необходимо создать стиль для определенного состояния элемента (например, внешний вид посещенной ссылки либо вид ссылки при наведенном на нее курсоре), а также для элемента, который четко не обозначен в структуре страницы. Примером такого элемента может быть первый символ в абзаце или первая строка.

С помощью псевдоклассов в CSS можно устанавливать стиль для уже существующих элементов веб-страницы, который будет применяться в случае каких-то действий пользователя. Псевдоэлементы же отличаются тем, что могут определять стиль несуществующего содержимого, а также четко не обозначенных элементов.

### Популярные псевдоклассы CSS

* :link – этот псевдокласс задает стиль ссылкам, по которым пользователь еще не перешел;
* :visited – этот же, наоборот, применяет стиль к уже посещенным ссылкам;
* :hover – определяет стиль элемента, когда на него наведен курсор (может применяться не только к ссылкам);
* :active – задает стиль активной ссылке (то есть, в момент клика по ней);
* :focus – применяет стиль к элементу при фокусировке на нем (например, при установке курсора в строку поиска);
* :not() – этот полезный псевдокласс позволяет выбрать и стилизовать только те элементы, которые не содержат селектор, указанный в скобках.

### Популярные псевдоэлементы CSS

Если псевдоклассы записываются с одним двоеточием, то псевдоэлементы – с двумя. Это было внедрено в CSS3 для того, чтобы различать псевдоклассы и псевдоэлементы между собой. Однако раньше этой разницы не существовало и с псевдоэлементами использовалось одно двоеточие. Сейчас браузеры поддерживают оба варианта написания (но не для всех случаев). Рассмотрим некоторые псевдоэлементы:

* ::after – используется вместе со свойством content и позволяет вывести необходимые данные после содержимого элемента;
* ::before – выполняет похожую функцию, что и предыдущий, только выводит данные перед содержимым элемента;
* ::selection – этот псевдоэлемент распознается браузерами только при использовании двух двоеточий и позволяет установить цвет и фон для текста, который выделен пользователем;
* ::first-letter – используется для изменения стиля первого символа в тексте элемента;
* ::first-line – используется для изменения стиля первой строки текста элемента.

Пример использования псевдоэлементов:

blockquote::before {

content: "«";

}

blockquote::after {

content: "»";

}

blockquote::selection {

color: #C8F7C5;

background-color: #1E824C;

}