**JavaScript DOM**

**§1 BOM (Browser Object Model)**

Объект **window** представляет собой глобальный объект - окно браузера. Все глобальные JavaScript объекты, функции и переменные автоматически становятся частью объекта window.

Объектная модель браузера (Browser Object Model, BOM) – это дополнительные объекты, предоставляемые браузером (окружением), чтобы работать со всеми элементами браузера, кроме разметки.  
  
Глобальные переменные являются свойствами объекта **window**.

Глобальные функции являются методами объекта **window**.

**§2 Подключение JavaScript**

Чтобы код Javascript сработал в браузере пользователя, его необходимо вставить между тегами **<script>** и **</script>**.

<html>

<head>

<title>Пример JavaScript</title>

</head>

<script>

console.log("Hello!");

</script>

<body>

...

<script src="script.js"></script>

</body>

</html>

JavaScript можно подключить как внутри тега **head**, так и внутри тега **body**. Разница заключается в том, что сработает первым - код JavaScript или сначала загрузится html-контент страницы, а затем с ним будет работать JavaScript.  
  
В голове сайта обычно загружаются скрипты с функциями или внешние библиотеки, которые не используют html-контент. В теле сайта мы используем JavaScript для работы с html-кодом.

Так же, мы можем использовать тег **script** без атрибута **src** и писать внутри него javaScript-код. Если внутри тега **script** присутствует путь к файлу (**src**), то внутри тега код будет игнорироваться.

**§3 Всплывающие окна сообщений**

В JavaScript есть три вида всплывающих окон:

* alert() - предупреждение
* prompt() - ввод данных
* confirm() - подтверждение

#### 

#### Окно с предупреждением

Окно с предупреждением часто используется, если вы хотите довести некую информацию до пользователя.

Когда появляется окно с предупреждением, то, чтобы продолжить, пользователь должен нажать кнопку "OK".

alert("Я окно с предупреждением!");

### 

#### Окно с подтверждением

Окно с подтверждением часто используется, если необходимо, чтобы пользователь принял или отверг что-то.

Когда появляется окно с подтверждением, то, чтобы продолжить, пользователь должен нажать либо на кнопку "OK", либо на кнопку "Отмена". Если пользователь нажимает на кнопку "OK", то окно возвращает значение true. Если же пользователь нажимает на кнопку "Отмена", то окно возвращает значение false.

let txt;

if (confirm("Нажмите кнопку!")) {

txt = "Вы нажали OK!";

} else {

txt = "Вы нажали Отмена!";

}

console.log(txt);

### 

#### Окно с предложением ввода

Окно с предложением ввода часто используется, когда необходимо, чтобы пользователь перед входом на страницу ввел какое-то значение.

Когда появляется окно с предложением ввода, то, чтобы продолжить, пользователь должен ввести какое-нибудь значение и нажать на кнопку "OK" или "Отмена".

Если пользователь нажимает кнопку "OK", то окно возвращает введенное значение. Если же пользователь нажимает кнопку "Отмена", то окно возвращает значение **null**.

let person = prompt("Введите ваше имя", "John Doe");

let txt;

if (person == null || person == "") {

txt = "Пользователь отменил ввод.";

} else {

txt = "Привет " + person + "! Как дела?";

}

console.log(txt);

# Свойства и методы объекта Window

**§1 Размеры окна браузера**

Чтобы определить размер окна браузера, можно использовать два свойства.

Оба свойства возвращают размер в пикселях:

* **window.innerHeight** – внутренняя высота окна браузера (в пикселях)
* **window.innerWidth** – внутренняя ширина окна браузера (в пикселях)

Окно браузера (область просмотра) не включает панель инструментов и полосу прокрутки.

**§2 Объект Screen**

Объект window.**screen** содержит информацию об экране пользователя.

Объект window.screen можно использовать без префикса window.

Свойства:

* screen.width
* screen.height
* screen.availWidth
* screen.availHeight
* screen.colorDepth
* screen.pixelDepth

Свойство screen.**width** возвращает ширину экрана пользователя в пикселях

Свойство screen.**height** возвращает высоту экрана пользователя в пикселях

Свойство screen.**availWidth** возвращает ширину экрана пользователя в пикселях минус интерфейс вроде таскбара Windows.

Свойство screen.**availHeight** возвращает высоту экрана пользователя в пикселях минус интерфейс вроде таскбара Windows.

Свойство screen.**colorDepth** возвращает число бит, используемых для отображения цвета.

Все современные компьютеры используют 24 или 32 аппаратных бита для цветового разрешения:

24 бита = 16 777 216 различных цветов - "True Colors"

32 бита = 4 294 967 296 различных цветов - "Deep Colors"

В более старых компьютерах использовалось 16 бит или 65 536 различных цветов - "High Colors".

В совсем старых компьютерах и мобильных телефонах использовалось 8 бит или 256 различных цветов - "VGA colors".

Значение #rrggbb (rgb), используемое в HTML, представляет палитру "True Colors" (16 777 216 цветов).

Свойство screen.**pixelDepth** возвращает глубину пикселя экрана в битах

В современных компьютерах глубина цвета и глубина пикселя — это одно и то же.

**§3 Объект Navigator**

Объект window.**navigator** содержит информацию о браузере посетителя страницы.

Объект window.navigator может записываться без префикса window.

Некоторые примеры:

* navigator.appName
* navigator.appCodeName
* navigator.platform

Свойство **cookieEnabled** возвращает значение true, если "cookie" (специальные файлы-метки) разрешены, в обратном случае возвращается значение false:

Свойство **appName** возвращает имя браузера, как приложения

Довольно странно, но именем приложения для браузеров IE11, Chrome, Firefox и Safari является "Netscape".

Свойство **appCodeName** возвращает кодовое имя браузера

Кодовым именем для браузеров Chrome, Firefox, IE, Safari и Opera является "Mozilla".

Свойство **product** возвращает имя движка браузера

Внимание! Не стоит полностью полагаться на данные этого свойства. Большинство браузеров возвращают имя "Gecko"!

Свойство **appVersion** возвращает информацию о версии браузера

Свойство **userAgent** возвращает заголовок пользовательского агента, посланного браузером серверу

Свойство **platform** возвращает платформу браузера (операционную систему)

Свойство **language** возвращает язык браузера:

Браузер подключен к сети Интернет?

Свойство **onLine** возвращает true, если браузер подключен к сети Интернет:

ВНИМАНИЕ!!!

* Информация из объекта navigator может вводить в заблуждение. Ее не следует использовать для определения версии браузера, так как:
* разные браузеры могут использовать одно и то же имя
* данные объекта navigator могут изменяться владельцем браузера
* некоторые браузеры дают неверные идентификаторы о себе, чтобы обходить проверку сайтов
* браузеры не могут сообщать о новых операционных системах, вышедших после самих браузеров

**§4 Объект History**

Объект window.**history** содержит историю посещенных браузером страниц.

Объект window.history может записываться без префикса window.

Чтобы обеспечить приватность пользователей, есть некоторые ограничения того, как JavaScript может получить доступ к этому объекту.

Свойство и методы объекта hidtory:

* history.**length** - количество страниц, открытых в одной и той же вкладке
* history.**back()** - то же самое, что нажатие на кнопку браузера "Назад"
* history.**forward()** - то же самое, что нажатие на кнопку браузера "Вперед"

Метод history.back() загружает предыдущий URL в списке посещенных страниц.

По сути этот метод действует так же, как кнопка браузера "Назад".

Метод history.forward() загружает следующий URL в списке посещенных страниц.

По сути этот метод действует так же, как кнопка браузера "Вперед".

**§5 Объект Location**

Объект window.**location** может использоваться для получения адреса (URL) текущей страницы и перенаправления браузера на новую страницу.

Свойства объекта location:

* location.href возвращает ссылку (URL) текущей страницы
* location.hostname возвращает доменное имя веб-хоста
* location.pathname возвращает путь и имя файла текущей страницы
* location.protocol возвращает использованный веб-протокол (http: или https:)
* location.port возвращает номер порта интернет хоста (текущей страницы).

**§6 Работа с окнами window**

#### **window.open(URL, имя, параметры)**- открывает новое окно window

Возвращает идентификатор нового окна для дальнейшей работы с ним

**URL** - является не обязательным аргументом. Указывает URL страницы, которая будет открыта в окне. Если данный параметр отсутствует окно будет открыто пустым.  
**имя** - является не обязательным аргументом. Указывает способ открытия или имя окна.

Возможные значения:

* *имя* - устанавливает имя окна
* *\_blank* - URL загружается в новом окне
* *\_parent* - URL загружается в родительском фрейме
* *\_self* URL - заменяет текущую страницу

**параметры** - является не обязательным аргументом.

Доступны следующие параметры:

* height устанавливает высоту окна в пикселях. Минимальное значение 100.
* location отвечает за отображение адресной строки (yes - отображать, no - неотображать).
* menubar URL отвечает за отображение верхнего меню (yes - отображать, no - неотображать).
* resizable отвечает за возможность изменения размеров окна (yes - можно изменять размер, no - нельзя изменять размер).
* scrollbars отвечает за отображения полосы прокрутки (yes - отображать, no - неотображать).
* width устанавливает ширину окна в пикселях. Минимальное значение 100.

#### **window.close()**- закрывает окно window

#### **window.focus()**- делает текущее окно активным

#### **window.blur()**- делает текущее окно неактивным

#### **window.moveTo(x, y)**- перемещает окно на указанную позицию

**x** - является обязательным параметром. Указывает координату, на которую будет перемещен верхний левый угол окна по оси x (по горизонтали).  
**y** - является обязательным параметром. Указывает координату, на которую будет перемещен верхний левый угол окна по оси y (по вертикали).

#### **window.moveBy(x, y)**- перемещает окно в любом направлении на указанное количество пикселей

**x** - является обязательным параметром. Смещение окна на указанное количество пикселей по оси x (по горизонтали).  
**y** - является обязательным параметром. Смещение окна на указанное количество пикселей по оси y (по вертикали).

#### **window.resizeTo(x, y)**- задает новые размеры окна

**x** - является обязательным параметром. Указывает новую ширину окна.  
**y** - является обязательным параметром. Указывает новую высоту окна.

#### **window.moveBy(x, y)**- изменяет размеры окна на указанное количество пикселей

**x** - является обязательным параметром. Изменение ширины окна на указанное количество пикселей.  
**y** - является обязательным параметром. Изменение высоты окна на указанное количество пикселей.

**§7 Создание и изменение нового документа**

**document** - объект window, который позволяет работать с html-контентом

Методы объекта **document**:

* document**.open()** - позволяет открыть новый документ для записи
* document**.write(str)** - добавляет на web-страницу текстовый контент
* document**.close()** - закрывает документ для записи

**§8 Таймеры**

Временные методы:

* **setTimeout(callback, time)** - выполняется через указанное количество миллисекунд (один раз).
* **setInterval(callback, time)** - выполняется через определенные промежутки времени (указанные в миллисекундах).
* **clearTimeout(id)** - отменяет запланированное методом setTimeout() выполнение кода.
* **clearInterval(id)** - прекращает повторное выполнение кода заданного setInterval().

**callback** - функция обратного вызова, используемая таймером в качестве аргумента

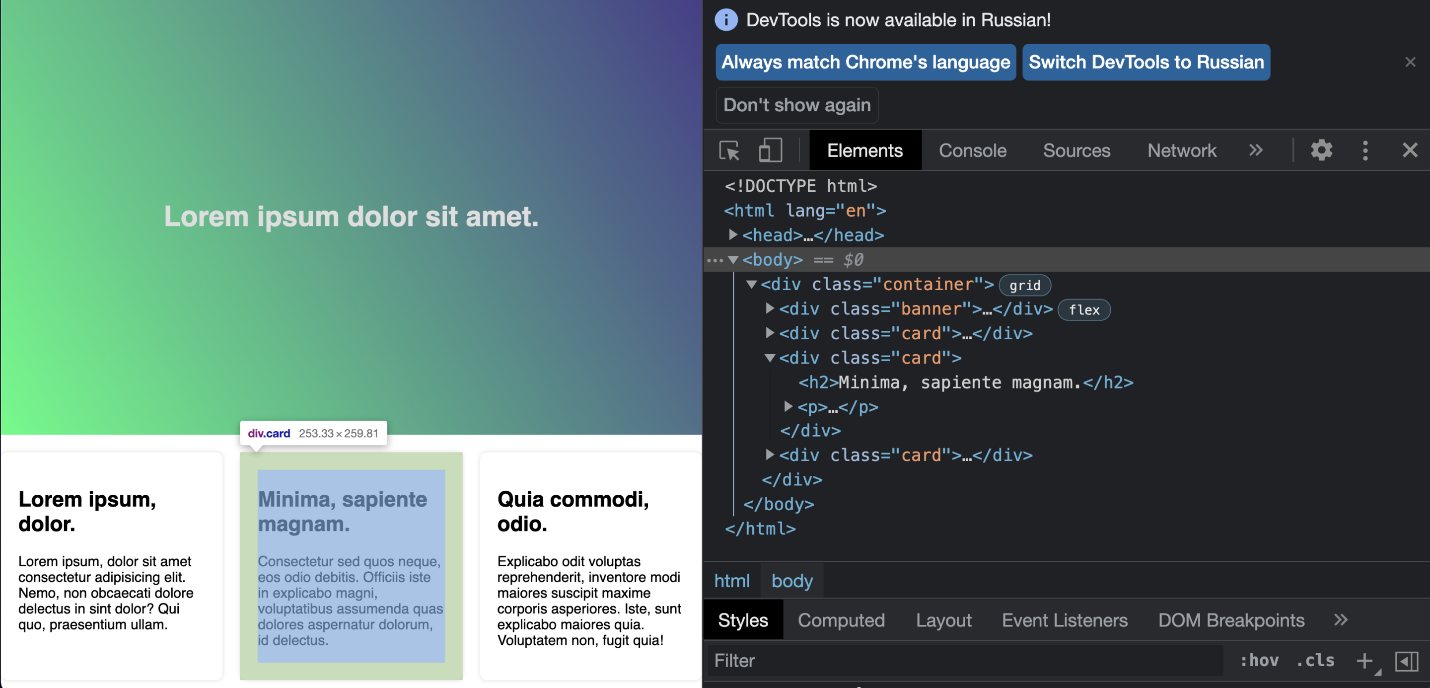
**time** - время, указанное в миллисекундах

**id** - идентификатор таймера в момент его вызова

# DOM, работа с узлами (Nodes)

**§1 Объект document**

Каждый раз, когда загружается HTML-страница, интерпретатор, находящийся внутри браузера, аккуратно собирает все содержимое HTML-файла при помощи JavaScript и показывает удобное представление HTML-кода в инструментах разработчика (**DevTools**) во вкладке **Elements**.



Благодаря этому, при наведении мышки на любой HTML-тег он подсвечивается в браузере, и мы видим информацию о нем — например, размеры у тега **div: 255.33 x 259.81**.

Такая структура в браузере называется **DOM-деревом**. Дерево, потому что можно развернуть каждый элемент HTML-кода и увидеть его вложенное содержимое, пока не придем к элементу, в котором уже нет ничего, кроме текста. Своего рода генеалогическое древо, но для HTML-элементов.

Одна из карточек (см. рис. выше) содержит в себе заголовок и абзац с текстом.

<div class="card">

<h2>Lorem ipsum, dolor.</h2>

<p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Nemo, non obcaecati dolore delectus in sint dolor? Qui quo, praesentium ullam.</p>

</div>

Тег **div** с классом **card** является родительским элементом, внутри которого находятся два дочерних тега **h2** и **p**. Тег **h2** является родительским элементом для текста, находящегося внутри него, как и тег **p** по отношению к своему вложенному тексту. Сами же теги **h2** и **p** называются братскими тегами (**siblings**) — у них общий родитель.

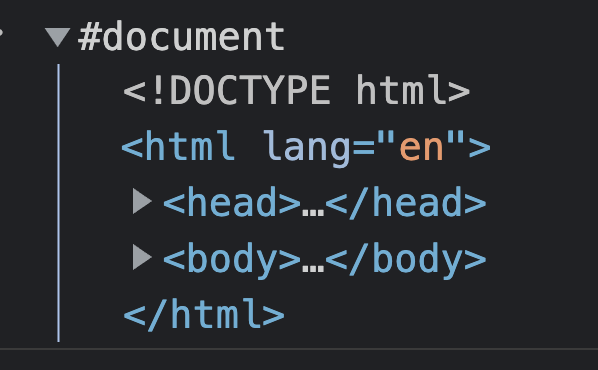
**DOM (Document Object Model)** — объектная модель документа. Каждый HTML-тег после преобразования в **DOM-дерево** становится объектом, и с ним можно работать так, как с обычным JavaScript-объектом.

У HTML-элемента есть определенные свойства и методы, благодаря которым можно динамически изменять контент, который видит пользователь при помощи JavaScript-кода.

**document** — специальный объект в браузере, который позволяет получить HTML-тег и начать работать с ним.

console.log(document);

Результат:



Объект **document** возвращает весь html-контент.

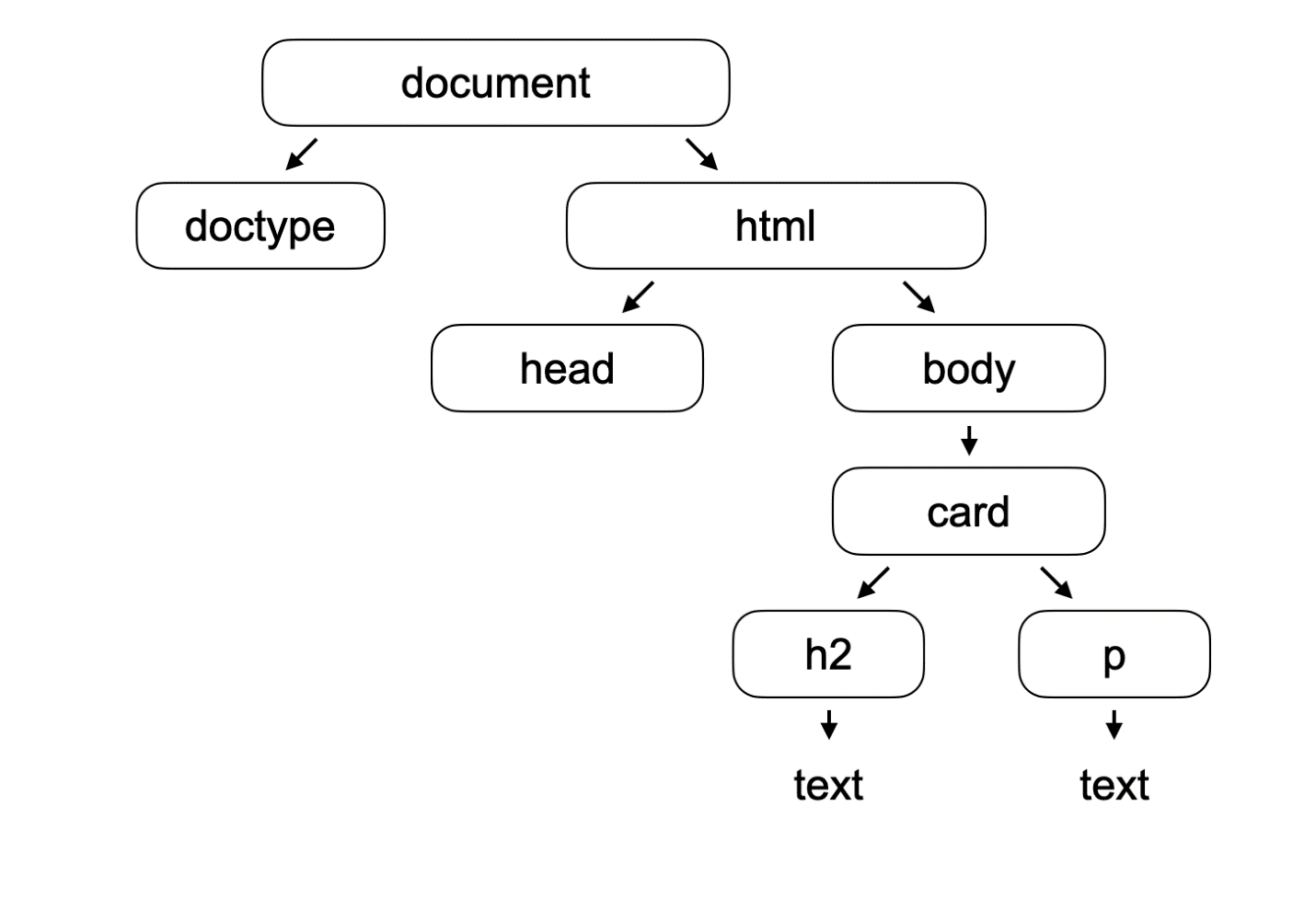
**§2 Структуры данных в DOM**

## Структуры данных в DOM

До этого мы уже столкнулись с фразой «чтобы найти нужный тег, JavaScript проверяет все ветки DOM-дерева, пока не доберется до искомого». Что это за ветки?

Они называются узлами (**node**). Каждая часть html-контента — узел:

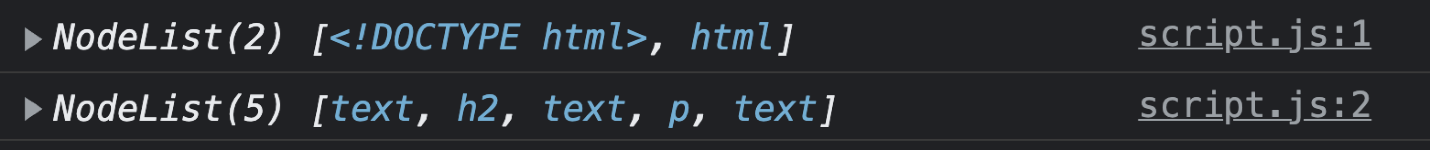
* Текст
* Тег
* Doctype



Для просмотра вложенных узлов используется свойство **childNodes**.

console.log(document.childNodes);

console.log(document.body.childNodes);



В результате вы увидите, что объект **document** имеет два дочерних узла — **doctype** и тег **HTML**, а элемент body — целых пять! Пять? «Но в body всего два тега?!» — спросите вы. Если бы мы писали HTML-код в одну строку, их действительно было бы два. Но ваша цель — красивый и удобочитаемый код, в котором по всем канонам есть **Enter** и отступы. Последние читаются **DOM-деревом** как текстовый узел, который функционально не отличается от текстового контента, видимого пользователю.

Кстати, свойство**childNodes** тоже возвращает список узлов (**NodeList**), а значит, с этим списком тоже можно работать почти как с массивом.

У каждого узла есть три свойства:

* **nodeName,**
* **nodeType,**
* **nodeValue**.

**nodeType** — порядковый номер типа узла. Это свойство помогает понять, как можно работать с узлом, а как нельзя. Например, текстовый узел можно прочитать или изменить, но добавить HTML-тег в текстовый узел нельзя. Всего существует 12 типов узлов, правда, большинство из них устарело и не используется. Сейчас используются только пять типов узлов:

* 1 — узел элемента,
* 3 — текстовый узел,
* 8 — узел комментария,
* 9 — узел документа (document),
* 10 — узел типа документа (doctype).

**nodeName** — название узла. Если вы возьмете текстовый узел и обратитесь к свойству **nodeName**, получится значение **#text**. Когда вы обращаетесь к этому же свойству узла элемента, например, тега **h2**, в результате получите текстовое названия тега, причем в верхнем регистре — **H2**.

**nodeValue** — свойство, которое отображает только внутреннее содержимое текстового узла, а именно: сам текст. При необходимости обратиться к нему из узла элемента или документа, вы получите **null**.

Узел элемента представляет собой класс **HTMLElement**. При помощи этого класса все html-теги, находящиеся в DOM-дереве, имеют общие свойства, такие как глобальные html-атрибуты**id, name, style**.

Каждый известный нам html-тег описан своим собственным классом, наследуемым от **HTMLElement**:

* **HTMLDivElement — div,**
* **HTMLAnchorElement — a,**
* **HTMLImageElement — img.**

Перечислять здесь все классы бессмысленно — их невероятно много. Каждый добавляет узлам собственные свойства: у тега a появляется возможность переходить по ссылке, а тег **img** отображается как изображение.

**§3 Перемещение между узлами**

Существует только одно свойство объекта **document**, которое позволяет обратиться на прямую к элементу **body**:

document.body

В остальных случаях, помимо перемещения между узлами при помощи списка childNodes, для поиска нужного узла существуют существуют следующие свойства:

* **parentNode**- позволяет обратиться к родительскому узлу
* **firstChild**- находит первый дочерний узел
* **lastChild** - обращается к последнему дочернему узлу
* **nextSibling**- позволяет обратиться к узлу, который находится после текущего (следующий)
* **previousSibling** - позволяет обратиться к узлу, который находится над текущим (предыдущий)

В большинстве случаев при работе с DOM-деревом, нам необходимо обращаться непосредственно к узлам элементов, игнорируя текстовые узлы (отступы, пробелы) или комментарии. Для этого имеется похожий набор свойств, который позволяет обратиться исключительно к узлу элемента:

* **parentElement**
* **firstElementChild**
* **lastElementChild**
* **nextElementSibling**
* **previousElementSibling**

# Изменение / добавление содержимого сайта при помощи JavaScript

**§1 Работа с содержимым DOM-узлов**

Теперь, когда вы умеете получать HTML-тег и понимаете структуру **DOM-дерева**, настало время поэкспериментировать и изменить контент.

Возьмем следующий контент

<div class="card" id="first">

<h2>Lorem ipsum, dolor.</h2>

<p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Nemo, non obcaecati dolore delectus in sint dolor? Qui quo, praesentium ullam.</p>

</div>

Есть переменная **card**, в которой хранится тег **div** с классом **card**, и переменная **caption**, в которой находится элемент **h2**.

Чтобы посмотреть, какой текст находится внутри тега **h2**, можно воспользоваться свойством **textContent**:

console.log(caption.textContent);

Lorem ipsum, dolor.

Однако, если вам потребуется применить это свойство к родительскому тегу **div**, увидите все текстовое содержимое тега.

console.log(card.textContent);

Lorem ipsum, dolor.

Lorem ipsum, dolor sit amet consectetur adipisicing elit. Nemo, non obcaecati dolore delectus in sint dolor? Qui quo, praesentium ullam.

Чего-то не хватает… Тегов! Свойство **textContent** соответствует получению только текста и ничему, кроме текста, кстати, со всеми отступами (табуляцией, пробелами и **Enter**).

Аналогично свойству **textContent** существует свойство **innerText**.

Чтобы получить текстовое содержимое заголовка, можете написать так:

console.log(caption.innerText);

Lorem ipsum, dolor.

А чтобы поменять текст внутри заголовка — вот так:

caption.innerText = "Это новый заголовок";

Разница между свойствами **innerText** и **textContent** незначительная:**innerText** определяется только для HTMLElement-объектов, а **textContent** — для всех узлов.

Итак, вы научились менять текстовое содержимое HTML-элемента, но что, если захочется изменить HTML-код внутри элемента?

Тут пригодится одно из самых любимых свойств frontend-разработчиков — **innerHTML**. В отличие от **innerText**, при использовании свойства **innerHTML** вы получите все содержимое элемента, включая теги, в строковом формате:

console.log(card.innerHTML);

<h2>Lorem ipsum, dolor.</h2>

<p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Nemo, non obcaecati dolore delectus in sint dolor? Qui quo, praesentium ullam.</p>

Соответственно, мы можем присвоить карточке новый html-контент, передав его в виде строкового типа данных.

card.innerHTML = "<h3>Маленький заголовок</h3><p>Тут будет текст…</p>";

При помощи оператора **+=** можно также добавить содержимое к карточке:

card.innerHTML += "<p>Еще один абзац с текстом</p>"