JS – Learn.JavaScript

Ссылки

Руководство по DOM, Мозила
https://developer.mozilla.org/ru/docs/DOM/DOM_Reference

Перечень событий в DOM
https://developer.mozilla.org/ru/docs/Web/Events

Спецификация DOM
Описывает структуру документа, манипуляции с контентом и события. Также называется DOM Living Standard:
https://dom.spec.whatwg.org

Спецификация CSSOM
Описывает файлы стилей, правила написания стилей и манипуляций с ними:
https://www.w3.org/TR/cssom-1/.

Спецификация HTML
Описывает язык HTML (например, теги) и BOM (объектную модель браузера) – разные функции браузера: setTimeout, alert, location и так далее:
https://html.spec.whatwg.org.

Интерфейсы Web API
Объекты для разработки веб-приложений называют интерфейсами.
https://developer.mozilla.org/ru/docs/Web/API

Chrome Developer Tools
Подробная документация по инструментам разработки в Хроме:
https://developers.google.com/web/tools/chrome-devtools.

Статья Html vs DOM
https://developers.google.com/web/tools/chrome-devtools/dom#appendix

Для поиска чего-либо обычно удобно использовать интернет-поиск со словами «WHATWG [термин]» или «MDN [термин]», например «whatwg localstorage», «mdn localstorage».

MDN HTML

Общий раздел по HTML с руководствами и остальными ссылками
Элементы HTML
Глобальные атрибуты элементов
Руководство по DOM
Перечень всех API

Формы
Все элементы форм тут
Руководство по HTML-формам
HTMLFormElement API
Проверка ограничений (валидация) форм: Constraint validation

Проекты

Игра с бегущим человеком
Отрабатываются:
- паралакс и скролл нескольких слоёв;
- события клика мыши, нажатия кнопки;
- таймеры (переключение уровней, скорость бега);
- установка звука;
- взаимодействие объектов (пересечения их границ);
- анимация простейшая;
- модальные окна;
- чекбоксы (мат вкл выкл);

Использовать:
<meter> - полоска прохождения уровня (не метр, а другую штуку)
<input> и <option> для выбора уровней и фонов.

Вопросы

Как сделать из коллекции массив?

```
Array.from(document.body.childNodes).filter
```

Чем отличается статическая коллекция от живой?
Живые коллекции всегда отражают текущее состояние документа и автоматически обновляются при его изменении. Важно, что они возвращают именно коллекцию, а не элемент, даже если элемент в ней один.

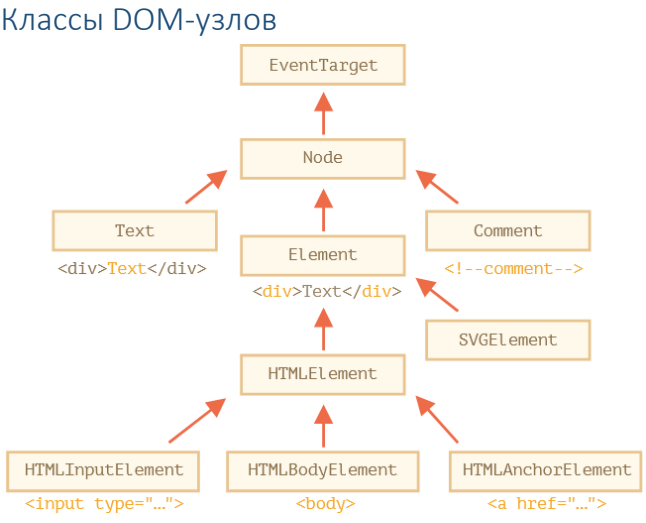
Как узнать класс DOM-узла?
Это обычные JavaScript объекты. Чтобы узнать класс, можно посмотреть конструктор, привести узел к строке или при помощи instance of:

```
alert( document.body.constructor.name );
// HTMLBodyElement

alert( document.body );
// [object HTMLBodyElement]

alert( document.body instanceof HTMLBodyElement ); // true
alert( document.body instanceof HTMLElement );     // true
// и т.д. по цепочке вверх
```

Классы DOM-узлов




```
EventTarget
это корневой «абстрактный» класс. Объекты этого класса никогда не создаются. Он служит
основой, благодаря которой все DOM-узлы поддерживают так называемые «события».

Node
```

```
также является «абстрактным» классом, и служит основой для DOM-узлов. Он обеспечивает
базовую функциональность: parentNode, nextSibling, childNodes и т.д. (это геттеры).
Объекты класса Node никогда не создаются. Но есть определённые классы узлов, которые
наследуют от него: Text – для текстовых узлов, Element – для узлов-элементов и Comment –
для узлов-комментариев.

Element
это базовый класс для DOM-элементов. Он обеспечивает навигацию на уровне элементов:
nextElementSibling, children и методы поиска: getElementsByTagName, querySelector. Браузер
поддерживает не только HTML, но также XML и SVG. Класс Element служит базой для следующих
классов: SVGElement, XMLElement и HTMLElement.

HTMLElement
является базовым классом для всех остальных HTML-элементов. От него наследуют конкретные
элементы:
HTMLInputElement – класс для тега <input>,
HTMLBodyElement – класс для тега <body>,
HTMLAnchorElement – класс для тега <a>,

и т.д, каждому тегу соответствует свой класс, который предоставляет определённые свойства
и методы.
```

Таким образом, полный набор свойств и методов данного узла собирается в результате наследования.
Например, DOM-объект для тега <input>. Он принадлежит классу HTMLInputElement.
Он получает свойства и методы из (в порядке наследования):

1. HTMLInputElement – этот класс предоставляет специфичные для элементов формы свойства,
2. HTMLElement – предоставляет общие для HTML-элементов методы (геттеры/сеттеры),
3. Element – предоставляет типовые методы элемента,
4. Node – предоставляет общие свойства DOM-узлов,
5. EventTarget – обеспечивает поддержку событий,
6. Object – методы «обычного объекта», такие как hasOwnProperty.

DOM-узлы – это обычные JavaScript объекты. Для наследования они используют классы, основанные на прототипах. Чтобы **узнать класс** DOM-узла, можно посмотреть конструктор, привести узел к строке или при помощи instance of:

```
alert( document.body.constructor.name );
// HTMLBodyElement

alert( document.body );
// [object HTMLBodyElement]

alert( document.body instanceof HTMLBodyElement ); // true
alert( document.body instanceof HTMLElement );     // true
// и т.д. по цепочке вверх
```

console.log(elem)	выводит элемент в виде DOM-дерева.
console.dir(elem)	выводит элемент в виде DOM-объекта, что удобно для анализа его свойств.

Node и Element

```
.childNodes           .children
.firstChild           .firstElementChild
.lastChild            .lastElementChild
.hasChildNodes()
.contains()
.childElementCount()

.nextSibling          .nextElementSibling
.previousSibling      .previousElementSibling
.parentNode           .parentElement

.nodeName             .tagName

.nodeValue            .innerHTML
.data                 .outerHTML
                      .textContent
```

CSS – селекторы

CSS-селектор это часть CSS-правила, которая позволяет указать, к какому элементу (элементам) применить стиль.
MDN

В качестве селекторов используются:

```
.class          класс
#id             id
tag             тег
[attribute]     по атрибуту
*               универсальный селектор

:hover          псевдокласс
:not(sltr)

::псевдоэлемент
```

теги

Стили будут применяться ко всем выбранным тегам.
Лучше так не делать и стилизовать через классы.

```
.html
<h1>Заголовок</h1>

.css
h1 {
свойство: значение;
свойство: значение;
}
```

.class

Class - глобальный атрибут, который используется для стилизации элементов.
Применяется к группе элементов, у которой прописан.
Стилизация элементов через class - хороший подход.

```
.html
<h1 class="title">Заголовок</h1>

.css
.title {
 свойство: значение;
 свойство: значение;
}
```

Если наименование класса состоит из нескольких слов, они записываются через дефис: первое-второе.
Через пробел можно присваивать тегу сразу несколько классов.

```
.html
<div class="flex-item green"></div>
<div class="flex-item red"></div>


.css
.flex-item {
  height: 100px;
  flex-grow: 1;
  flex-basis: 0;
}

.red { background-color: #e03333; }
.green { background-color: #83b10d; }
```

#id

id - глобальный атрибут, уникальный идентификатор элемента.
Нельзя делать 2 элемента с одним и тем же Id.
Стилизация через id – плохой подход. В практике такое не используется.

```
.html
<h1 id="title">Заголовок</h1>

.css
#title {
свойство: значение;
свойство: значение;
}
```

* звёздочка

Звездочка (*) - универсальный селектор для CSS, соответствует любому тегу. MDN
В звездочка (*) может использоваться в комбинации с пространством имён:

```
*::after
*::before

ns|* - вхождения всех элементов в пространстве имён ns #?
 *|* - находит все элементы
  |* - ищет все элементы без объявленного пространства имён
```

[ ] Селекторы атрибутов

Вообще, это селекторы CSS, но они также используются при поиске тех или иных элементов в DOM через querySelector. Можно задавать условия поиска.
MDN

```
[attr]
Выбрать элемент, у которого есть атрибут attr (с любым значением).

[attr="value"]
Выбрать элемент, у которого есть атрибут attr и он точно равнен value.

[attr~="value"]
Выбрать элемент, у которого есть атрибут attr, его значение – набор слов разделенных
пробелами, одно из которых равно value.

[attr|="value"]
Выбрать элемент, у которого есть атрибут attr, и его значение:
- или в точности равно "value"
- или может начинаться с "value" со сразу же следующим дефисом «-» U+002D.
Это может быть использовано когда язык описывается с подкодом.

[attr^="value"]
Выбрать элемент, у которого есть атрибут attr, его значение начинается с "value"

[attr$="value"]
Выбрать элемент, у которого есть атрибут attr, его значение заканчивается на "value"

[attr*="value"]
Значение содержит по крайней мере одно вхождение строки "value" как подстроки.
```

Можно комбинировать селекторы, указывая их один за другим.
В примере ниже скомбинирован селектор атрибута и селектор всевдокласса.
Выбрать все ссылки, href которых содержит «://», но которые не начинаются с «http://internal.com»:

```
selector = 'a[href*="://"]:not([href^="http://internal.com"])'
```

Ещё примеры:

```
/* Все span с атрибутом "lang" будут жирными */
span[lang] {font-weight:bold;}

/* Все span в Португалии будут зелеными */
span[lang="pt"] {color:green;}

/* Все span с американским английским будут синими */
span[lang~="en-us"] {color: blue;}

/* Любые span на китайском языке будут красными, как на упрощенном китайском (zh-CN) так и
на традиционном (zh-TW) */
span[lang|="zh"] {color: red;}

/* Все внутренние ссылки будут иметь золотой фон */
a[href^="#"] {background-color:gold}

/* Все ссылки с url заканчивающимся на .cn будут красными */
a[href$=".cn"] {color: red;}

/* Все ссылки содержащие "example" в url будут иметь серый фон */
a[href*="example"] {background-color: #CCCCCC;}
```

: Псевдоклассы

Псевдокласс в CSS определяет состояние элемента или его положение в dom-дереве.
Например, :hover или :first-child.
Список стандартных псевдоклассов в документации MDN.

```
selector:pseudo-class {
  property: value;
}

div:hover {
  background-color: #F89B4D;
}
```

Некоторые псевдоклассы:

```
:link       элементы, которые являются ссылками на странице
:active     ссылки, на которые сейчас кликнули
:visited    ссылки, по которым уже переходили
:hover      на которые наведён курсор
:focus      на которых сфокусировались с клавиатуры

:root выбирает корневой элемент HTML-документа

:first-child только первый элемент
:last-child  только последний элемент
:only-child  элемент, который является единственным дочерним для своего родительского

:nth-child
:nth-child(3n)    каждый третий
:nth-child(n+2)   все, начиная со второго
:nth-child(odd)   все нечётные элементы
:nth-child(even)  все чётные элементы

:first-of-type
:last-of-type
:only-of-type   элемент, который является единственным элементом данного типа.

:empty          выбирает пустые элементы.

:not(selector)  выбрать все элементы, не соответствующие селектору
```

Примеры:

```
ul li:nth-child(3n)   каждый третий li
ul li:nth-child(n+2)  все li, начиная со второго
ul li:nth-child(odd)  обратиться ко всем нечётным элементам
ul li:nth-child(even) обратиться к чётным элементам
ul li:first-child     только первый элемент
```

```
:active
:any
:any-link
:checked
:default
:defined
:dir()
:disabled
:empty
:enabled
:first
:first-child
:first-of-type
:fullscreen
:focus
:hover
:indeterminate
:in-range
:invalid
:lang()
:last-child
:last-of-type
:left
:link
:not()
:nth-child()
:nth-last-child()
:nth-last-of-type()
:nth-of-type()
:only-child
:only-of-type
:optional
:out-of-range
:read-only
:read-write
:required
:right
:root
:scope (en-US)
:target
:valid
:visited
```

:not()

Псевдокласс-функция :not(X) принимает простой селектор X в качестве аргумента. Он находит элементы, не соответствующие селектору. *X* не должен содержать других отрицательных селекторов.
MDN

```
:not(selector) { style }

li:not(first-child)
выберутся все элементы списка, которые не являются первым ребёнком.

Пример
Выбрать все ссылки с атрибутом «href» которые:
- содержит «://»
- не начинается с «http://internal.com»

[attr*=value] - значение атрибута содержит вхождение строки "value" как подстроки.
[attr^=value] - значение атрибута начинается с "value"

selector = 'a[href*="://"]:not([href^="http://internal.com"])'
```

:: Псевдоэлементы

Псевдоэлемент в CSS позволяет стилизовать определённую часть выбранного элемента.
Например, ::first-line.
Следует использовать двойное двоеточие (::) вместо одинарного. Однако, большинство браузеров поддерживают оба синтаксиса для псевдоэлементов.
Список псевдоэлементов в документации. MDN

В селекторе можно использовать только один псевдоэлемент. Он должен находиться после простых селекторов в выражении.

```
selector::pseudo-element {
  property: value;
}
```

Список стандартных псевдоэлементов из документации:

```
::after
::before
::cue
::first-letter   выделяет первую букву текста
::first-line     выделяет первую строку текста
::selection      часть документа, выделенная пользователем
::slotted
::backdrop
```

```
::placeholder
::marker
::spelling-error (en-US)
::grammar-error
```

Комбинаторы селекторов

```
Комбинаторы на MDN

class1, class2    правила для нескольких независимых классов
class1.class2     элемент, в котором присутствуют одновременно 2 класса

div + div         выбрать следующего соседа (брата) на одном уровне вложенности
div ~ div         выбрать справа всех соседей (братьев) на одном уровне вложенности

sltr1 > sltr2     выбрать прямых потомков с таким селектором
sltr1 >> sltr2    выбрать потомков любого уровня вложенности
sltr1 sltr2       чаще записывается так


MDN для:
~
>
>>
```

Полный перечень команд

```
Верхушка
<html> = document.documentElement
<head> = document.head
<body> = document.body


Поиск элементов
document.getElementById('id')

elem.querySelectorAll(css)
elem.querySelector(css)

elem.getElementsByClassName(className)
document.getElementsByName(name)
elem.getElementsByTagName(tag)    если поставить * вместо тега, то вернёт всех потомков


Навигация
node
.childNodes
.firstChild
.lastChild
.hasChildNodes() функция проверки дочерних узлов

.nextSibling
.previousSibling.

.parentNode

element
.children
.firstElementChild
.lastElementChild
.nextElementSibling
.previousElementSibling.

.parentElement


Проверка соответствия
elem.closest(css)       ближайший соответствующий указанному правилу предок
elem.matches(css)       проверяет, удовлетворяет ли elem указанному CSS-селектору
lemA.contains(elemB)    проверяет на наличие потомка внутри
node.hasChildNodes()
node.childElementCount


Имена и содержимое
Имя тега или узла
.nodeName  название тега или тип узла
.tagName   название тега, есть только у элементов Element
.nodeType  старомодный способ узнать тип узла

Класс узла
Можно узнать API таким образом
document.body.constructor.name  HTMLBodyElement
document.body.toString;         [object HTMLBodyElement]

Содержимое узлов
.nodeValue    текст внутри ноды. Можно менять. См. «Как вытащить текст из узлов»
.data         текст внутри ноды или элемента. Можно менять.

Содержимое элементов
.innerHTML    внутреннее HTML-содержимое узла-элемента. Можно менять.
.outerHTML    запись в elem.outerHTML не меняет elem. Можно менять.
.innerText    текстовое содержимое элемента и его потомков
.textContent  текст внутри узла-элемента (вычет всех тегов). Можно менять.
.hidden       true делает элемент невидимым, как style="display:none"

node.childElementCount  кол-во вложенных элементов узла

Встроенные свойства
.value   значение для <input>, <select> и <textarea> (HTMLInputElement, HTMLSelectElement…)
.href    адрес ссылки «href» для <a href="..."> (HTMLAnchorElement)
.id      значение атрибута «id» для всех элементов (HTMLElement)


HTML-атрибуты
elem.attributes                    получить сразу все атрибуты элемента как коллекцию
elem.hasAttribute('name')          проверяет наличие атрибута.
elem.getAttribute('name')          получает значение атрибута.
elem.setAttribute('name', value)   устанавливает значение атрибута.
elem.removeAttribute('name')       удаляет атрибут

Запись нестандартных свойств в HTML
<body data-order-state="ok">

Получение нестандартных свойств в DOM
document.body.dataset.orderState


Изменение DOM
Создать узел
document.createElement('element') let div = document.createElement('div');
document.createTextNode('text')   let textNode = document.createTextNode('А вот и я');

Копирование узлов
```

```
elem.cloneNode(true)   клон элемента со всеми атрибутами и дочерними элементами
elem.cloneNode(false)  клон без дочерних элементов

Замена узлов
node.replaceWith(...nodes or strings)    заменяет node заданными узлами или строками

Удаление узлов
node.remove()


Безопасная вставка узлов и строк
Можно создать элемент, сохранить в переменной и вставить как ноду, но не как код
node.append(...nodes or strings)    добавляет узлы или строки в конец node
node.prepend(...nodes or strings)   вставляет узлы или строки в начало node
node.before(...nodes or strings)    вставляет узлы или строки до node
node.after(...nodes or strings)     вставляет узлы или строки после node


Вставка HTML кода (элементов) и его братья
elem.insertAdjacentHTML(where, html)     универсальный метод по вставке кода html
elem.insertAdjacentText(where, text)     вставить текст
elem.insertAdjacentElement(where, elem)  вставить готовую ноду

where
"afterbegin"    вставить html в начало elem
"beforeend"     вставить html в конец elem
"beforebegin"   вставить html до elem
"afterend"      вставить html после elem


html
строка, которая будет вставлена как код HTML


Устаревшие методы
parentElem.appendChild(node)
добавляет node в конец дочерних элементов parentElem

parentElem.insertBefore(node, nextSibling)
вставляет node перед nextSibling в parentElem

parentElem.replaceChild(node, oldChild)
заменяет oldChild на node среди дочерних элементов parentElem

parentElem.removeChild(node)
удаляет node из parentElem (предполагается, что он родитель node).

let fragment = new DocumentFragment()
document.write('<b>Привет</b>');


Классы
elem.className  получить или записать атрибут элемента class="…" в виде строки

elem.classList              специальный объект для добавления/удаления одного класса
add('class1', 'class2')     Добавляет элементу указанные классы
remove('class1', 'class2')  Удаляет у элемента указанные классы
item(Number)                Результат аналогичен вызову classList[index]
length                      кол-во классов у элемента
contains('class1')    Проверяет, есть ли данный класс у элемента (вернет true или false)
toggle('class1' [,Boolean]) Если класс у элемента отсутствует - добавляет, если уже
присутствует - убирает. Если вторым параметром передан false - удаляет указанный класс, а если
true - добавляет.

Стили
elem.style.property         объект, чьи свойства – правила, записанные в атрибуте эл-та
elem.style.color = 'red'
elem.style.cssText          переписать или получить весь атрибут style
window.getComputedStyle(el) живой объект со сложенными стилями


Размеры и прокрутка элемента
.offsetParent    ссылка на предка элемента
.offsetTop       расстояние вверх до родителя
.offsetLeft      расстояние слева от родителя

.offsetWidth     ширина + padding + border
.offsetHeight    высота + padding + border

.clientTop       толщина рамки border (чаще всего)
.clientLeft

.clientWidth     ширина + padding
.clientHeight    высота + paddind

.scrollWidth     вся ширина элемента на странице, в т.ч. с невидимой частью
.scrollHeight    вся высота элемента на странице, в т.ч. с невидимой частью

.scrollLeft      ширина невидимой, уже прокрученной части элемента (можно записывать)
.scrollTop       высота невидимой, уже прокрученной части элемента (можно записывать)

let scrollBottom = elem.scrollHeight - elem.scrollTop - elem.clientHeight;
узнать размер прокрутки снизу (сколько ещё крутить осталось)

Высота всего documentElement
let height = Math.max(
  document.body.scrollHeight, document.documentElement.scrollHeight,
  document.body.offsetHeight, document.documentElement.offsetHeight,
  document.body.clientHeight, document.documentElement.clientHeight
);



Размеры и прокрутка окна
Размеры окна
document.documentElement.clientWidth   ширина окна браузера (без полосы прокрутки)
document.documentElement.clientHeight  высота окна браузера (без полосы прокрутки)

window.innerWidth                  ширина окна браузера (включая полосу прокрутки)
window.innerHeight                 высота окна браузера (включая полосу прокрутки)

Прокрутка окна и элементов
window.pageYOffset                 показать текущую прокрутку по вертикали (только чтение)
window.pageXOffset                 показать текущую прокрутку по горизонтали (можно без window)

documentElement.scrollLeft         показать или установить прокрутку (окно и элементы)
documentElement.scrollTop          показать или установить прокрутку (окно и элементы)

window.scrollBy(x, y)              прокрутка относительно текущего положения на n пиксесей
window.scrollTo(pageX, pageY)      прокрутка на абсолютные координаты

e.scrollIntoView(true)             прокрутить к elem (появится в верху)
e.scrollIntoView(false)            прокрутить к elem (появится внизу)

старые способы прокрутить что-то
document.documentElement.scrollTop
document.documentElement.scrollLeft
window.scrollByLines
window.scrollByPages

document.body.style.overflow = "hidden"   отключить прокрутку
document.body.style.overflow = ""         включить обратно
```

Координаты

Относительно окна

```
let el = document.elementFromPoint(x, y)
вернуть элемент, который находится по этим координатам


domRect = el.getBoundingClientRect()
domRect.x        X-координата начала прямоугольника относительно окна
domRect.y        Y-координата начала прямоугольника относительно окна

domRect.width    ширина прямоугольника (м.б. отрицательным)
domRect.height   высота прямоугольника (м.б. отрицательным)

domRect.top      Y-координата верхней границы прямоугольника (доп. зависимое свойство)
domRect.bottom   Y-координата нижней границы прямоугольника (доп. зависимое свойство)

domRect.left     X-координата левой границы прямоугольника (доп. зависимое свойство)
domRect.right    X-координата правой границы прямоугольника (доп. зависимое свойство)
```

Относительно документа надо вычислять

```
pageY = clientY + высота вертикально прокрученной части документа
pageX = clientX + ширина горизонтально прокрученной части документа

Функция вычисления координат объекта относительно документа
function getCoords(elem) {
  let box = elem.getBoundingClientRect();
  return {
    top: box.top + pageYOffset,
    left: box.left + pageXOffset
  };
}
```

События

```
document.addEventListener("DOMContentLoaded", function() { код });
выполнится только после полной загрузки документа
```

Повесить обработчик

```
<e onclick="alert('Клик!')">   повесить обработчик через html
e.onclick = sayThanks;         повесить обрабочик через dom-свойство
e.onclick = null               удалить обработчик


element.addEventListener('event', handler [, options]);  лучший способ вешать обработчики
element.removeEventListener('event', handler[, options]);  удалить обработчик
event
Имя события без on, например "click"
handler
Ссылка на функцию-обработчик
Options {           объект со свойствами
  once: true,       обработчик автоматом удалится после выполнения
  capture: false,   фаза, на которой сработает обработчик. Если true – сработает на погружении.
  passive: true     обработчик никогда не вызовет preventDefault()
}
```

Всплытие

```
event.target         ссылка на объект, который был инициатором события
event.currentTarget  ссылка на элемент, в котором в данный момент обрабатывается событие (this)

event.stopPropagation()            выполняет текущую обработку события и останавливает всплытие
event.stopImmediatePropagation();  останавливает и текущую обработку события, и всплытие

event.eventPhase     номер фазы, на которой событие было поймано
```

Действия браузера по умолчанию

```
event.preventDefault()  отмена действия браузера в eventListener
event.defaultPrevented  будет равняться true, если действие по умолчанию было отменено

onclick="return false"  отмена действия браузера, если событие назначено через 'on'
```

Генерация пользовательских событий

Основы событий мыши

MouseEvent API

События мыши

```
mouseover     курсор над элементом
mouseout      курсор ушёл с элемента

mouseenter    курсор над элементом или его потомками. Не всплывает.
mouseleave    курсор ушёл с элемента или его потомков. Не всплывает.

mousemove     движение мыши над элементом

click         клик на элементе левой кнопкой или тап на сенсорном экране. mousedown + mouseup
dblclick      двойной клик
contextmenu   открытие контекстного меню (пкм обычно, это не совсем событие мыши)
oncopy        запрет копирования, если запретить действие браузера по умолчанию.
```

Другие свойства

```
me.which      возвращает код нажатой клавиши мыши
```

Координаты курсора мыши

```
me.clientX    X относительно экрана
me.clientY    Y относительно экрана

me.pageX      X относительно всего документа
me.pageY      Y  относительно всего документа
```

Движение мыши

```
me.relatedTarget  второстепенная цель некоторых событий
для mouseover     элемент, с которого курсор ушёл
для mouseout      элемент, на который курсор перешёл
```

Drag'n'Drop с мышью

Drag and drop в интерфейсах веб API

Клавиши-модификаторы

```
ev.altKey     true/false
ev.ctrlKey    true/false
ev.metaKey    true/false
ev.metaKey    true/false
```

keydown и keyup

```
keydown    событие при нажатии клавиши
keyup      событие при отпускании клавижи

свойства
ke.code    код нажатой клавиши, начинается с заглавной буквы
ke.key     символ, введённый с клавиатуры
ke.repeat  true/false, была ли клавиша зажата (автоповтор)
```

Pointer events, события указателя

API PointerEvent

События Pointer Events
pointerdown
pointerup
pointermove
pointerover
pointerout
pointerenter
pointerleave

pointercancel
gotpointercapture
lostpointercapture

Свойства Pointer Events
Все свойства MouseEvent + дополнительные свойства здесь.
pe.pointerId уникальный идентификатор указателя, вызвавшего событие (м.б. несколько)
pe.pointerType тип указывающего устройства: «mouse», «pen» или «touch»
pe.isPrimary true для основного указателя (первый палец в мульти-тач)

pe.width ширина области соприкосновения указателя (пальца). Если не поддерживается: 1
pe.height высота области соприкосновения указателя. Если не поддерживается: 1
pe.pressure степень давления указателя от 0 до 1. Если не поддерживают: 0.5 либо 0

pe.tiltX специфичные для пера свойства
pe.tiltY
pe.twist

t.tangentialPressure нормализованное тангенциальное давление

Смежные:
el.setPointerCapture(pointerId) привязывает события с данным pointerId к elem
el.releasePointerCapture()
touch-action: none прописывается в CSS для отмены действий по умолчанию

Прокрутка
window.pageYOffset показать кол-во пикселей, на которое прокручено окно по Y. Только чтение
window.pageXOffset показать кол-во пикселей, на которое прокручено окно по X. Только чтение

window.scroll(x, y) прокрутить страницу до указанного места
window.scrollTo(x, y) тот же самый эффект
window.scrollBy(x, y) прокручивает документ на указанные величины

window.scrollByPages() Прокручивает документ на указанное число страниц
window.scrollByLines() Прокручивает документ на заданное число строк

overflow: hidden правило CSS запрещает прокрутку

Событие
scroll событие возникает при прокрутке области просмотра документа или элемента
onscroll

Формы, элементы управления
Формы
document.forms возвращает коллекцию форм в текущем документе. Доступны по имени или индексу
form.elements возвращает коллекцию элементов внутри формы (вне зависимости от вложенности)
form.elemName сокращённое обращение к элементу внутри формы
element.form обратная ссылка элемента на форму

<input>
input.value задать или получить значение input
input.checked true/false означает наличие или отсутствие выбора у radio или checkbox

<textarea>
textarea.value задать или получить значение

<select>
select.options коллекция из подэлементов <option>
select.value = 'banana' значение выбранного в данный момент <option> (чтение и запись)
select.selectedIndex = n индекс выбранного <option>
option.selected см. ниже

<option>
option.selected true/false, выбрана ли эта опция.
option.index индекс опции среди других в списке <select>.
option.text содержимое опции (то, что видит посетитель).

new Option(text, value, defaultSelected, selected);
text текст внутри <option>,
value значение
defaultSelected если true, то ставится HTML-атрибут selected,
selected если true, то элемент <option> будет выбранным.

Фокусировка
SelectElement.autofocus = Bool установить автофокус на элементе

События
focus событие вызывается в момент фокусировки. Не всплывает.
blur событие вызывается в момент потери фокуса. Не всплывает.
focusin всплывает
focusout всплывает

Методы
HTMLElement.focus() установить фокус на элементе
HTMLElement.blur() снять фокус с элемента

tabindex='' число, порядок последовательной навигации (глобальный html-атрибут)
el.tabIndex dom-свойство, порядковый номер фокусировки

Document.activeElement вернёт текущий сфокусированный элемент
el.autofocus HTML-атрибут устанавливает фокус на элемент, когда страница загружается

Обновление данных
HTMLElement: change событие срабатывает по окончании изменения элемента
HTMLElement: input срабатывает каждый раз при изменении значения

Element: cut при вырезании
Element: copy при копировании
Element: paste при вставке

ClipboardEvent API; события, связанные с изменением буфера обмена: cut, copy и paste.
.clipboardData трансфер-объект с данными при копировании, вырезании или вставке.

DataTransfer API;
.getData(format) Возвращает данные для указанного типа

Отправка формы
HTMLFormElement API
.submit() отправляет форму <form>, но не инициализирует событие submit
.requestSubmit() отправляет форму, как если бы была нажата кнопка «submit»
submit event событие

API

API Window

```
Window.alert()
Displays an alert dialog.
Window.prompt()
Returns the text entered by the user in a prompt dialog.
Window.confirm()


Window.innerHeight
Gets the height of the content area of the browser window including, if rendered, the
horizontal scrollbar.
Window.innerWidth
Gets the width of the content area of the browser window including, if rendered, the vertical
scrollbar.


Window.scroll()
Scrolls the window to a particular place in the document.
Window.scrollBy()
Scrolls the document in the window by the given amount.
Window.scrollTo()
Scrolls to a particular set of coordinates in the document.


Element.scrollLeft
Is a Number representing the left scroll offset of the element.
Element.scrollTop
A Number representing number of pixels the top of the document is scrolled vertically.

Window.pageXOffset Read only
An alias for window.scrollX.
Window.pageYOffset Read only
An alias for window.scrollY


Window.location
Gets/sets the location, or current URL, of the window object.
```

API Document

This interface also inherits from the Node and EventTarget interfaces.
Некоторые свойства:

```
API Document =>
Properties

Document.activeElement
Returns the Element that currently has focus.
Document.childElementCount
Returns the number of child elements of the current document

Доступ к базовым тегам
Document.head
Returns the <head> element of the current document.
Document.body
Returns the <body> or <frameset> node of the current document.
Document.documentElement
Returns the Element that is a direct child of the document. For HTML documents, this is
normally the HTMLHtmlElement object representing the document's <html> element.

Доступ к коллекциям
Document.forms
Returns a list of the <form> elements within the current document.
Document.images
Returns a list of the images in the current document.
Document.links
Returns a list of all the hyperlinks in the document.


XЗ
Document.characterSet
Returns the character set being used by the document.
Document.compatMode
Indicates whether the document is rendered in quirks or strict mode.
Document.contentType
Returns the Content-Type from the MIME Header of the current document.
Document.doctype
Returns the Document Type Definition (DTD) of the current document.
Document.documentURI
Returns the document location as a string.
Document.embeds
Returns a list of the embedded <embed> elements within the current document.
Document.fonts
Returns the FontFaceSet interface of the current document.
Document.fullscreenElement Read only
The element that's currently in full screen mode for this document.
Document.hidden
Returns a Boolean value indicating if the page is considered hidden or not.
Document.implementationRead only
Returns the DOM implementation associated with the current document.
Document.mozSyntheticDocument
Returns a Boolean that is true only if this document is synthetic, such as a standalone image,
video, audio file, or the like.
Document.pictureInPictureElement Read only
Returns the Element currently being presented in picture-in-picture mode in this document.
Document.pictureInPictureEnabled Read only
Returns true if the picture-in-picture feature is enabled.
Document.pluginsRead only
Returns a list of the available plugins.
Document.pointerLockElement Read only
Returns the element set as the target for mouse events while the pointer is locked. null if
lock is pending, pointer is unlocked, or if the target is in another document.
Document.featurePolicy
Read only
Returns the FeaturePolicy interface which provides a simple API for introspecting the feature
policies applied to a specific document.
Document.scriptsRead only
Returns all the <script> elements on the document.
Document.scrollingElementRead only
Returns a reference to the Element that scrolls the document.
Document.styleSheets Read only
Returns a StyleSheetList of CSSStyleSheet objects for stylesheets explicitly linked into, or
embedded in a document.
Document.timeline
Read only
Returns timeline as a special instance of DocumentTimeline that is automatically created on
page load.
Document.visibilityState
Returns a string denoting the visibility state of the document. Possible values are visible,
hidden, prerender, and unloaded.

Extensions for HTMLDocument
Event handlers


API Document =>
Methods (некоторые)
Создание
```

Document.createAttribute()
Creates a new Attr object and returns it.
Document.createComment()
Creates a new comment node and returns it.
Document.createElement()
Creates a new element with the given tag name.
Document.createTextNode()
Creates a text node.

Получение элементов из точки
Document.elementFromPoint()
Returns the topmost element at the specified coordinates.
Document.elementsFromPoint()
Returns an array of all elements at the specified coordinates.

Extension for HTML documents (методы)
Document.write()
Writes text in a document.

API Node

Некоторые свойства:

Node.childNodes
Returns a live NodeList containing all the children of this node (including elements, text and comments). NodeList being live means that if the children of the Node change, the NodeList object is automatically updated.
Node.firstChild
Returns a Node representing the first direct child node of the node, or null if the node has no child.
Node.lastChild
Returns a Node representing the last direct child node of the node, or null if the node has no child.

Node.parentNode
Returns a Node that is the parent of this node. If there is no such node, like if this node is the top of the tree or if doesn't participate in a tree, this property returns null.
Node.parentElement
Returns an Element that is the parent of this node. If the node has no parent, or if that parent is not an Element, this property returns null.

Node.nextSibling
Returns a Node representing the next node in the tree, or null if there isn't such node.
Node.previousSibling
Returns a Node representing the previous node in the tree, or null if there isn't such node.

Node.nodeName
Returns a DOMString containing the name of the Node. The structure of the name will differ with the node type. E.g. An HTMLElement will contain the name of the corresponding tag, like 'audio' for an HTMLAudioElement, a Text node will have the '#text' string, or a Document node will have the '#document' string.
Node.nodeType
Returns an unsigned short representing the type of the node. Possible values are:
Node.nodeValue
Returns / Sets the value of the current node.

Node.textContent
Returns / Sets the textual content of an element and all its descendants.

API Node => Methods (некоторые)
Копирование, вставка узлов
Node.appendChild(childNode)
Adds the specified childNode argument as the last child to the current node.
If the argument referenced an existing node on the DOM tree, the node will be detached from its current position and attached at the new position.
Node.insertBefore()
Inserts a Node before the reference node as a child of a specified parent node.

Node.cloneNode()
Clone a Node, and optionally, all of its contents. By default, it clones the content of the node.

Проверка на содержание ноды
Node.contains()
Returns a Boolean value indicating whether or not a node is a descendant of the calling node.

Node.hasChildNodes()
Returns a Boolean indicating whether or not the element has any child nodes.

API ParentNode

Некоторые свойства

ParentNode.childElementCount
Returns the number of children of this ParentNode which are elements.

Получение дочерних узлов-элементов
ParentNode.children
Returns a live HTMLCollection containing all of the Element objects that are children of this ParentNode, omitting all of its non-element nodes.
ParentNode.firstElementChild
Returns the first node which is both a child of this ParentNode and is also an Element, or null if there is none.
ParentNode.lastElementChild
Returns the last node which is both a child of this ParentNode and is an Element, or null if there is none.

Вставка узлов
ParentNode.append()
Inserts a set of Node objects or DOMString objects after the last child of the ParentNode. DOMString objects are inserted as equivalent Text nodes.
ParentNode.prepend()
Inserts a set of Node objects or DOMString objects before the first child of the ParentNode. DOMString objects are inserted as equivalent Text nodes.

API ChildNode

Удаление и замена
ChildNode.remove()
Removes this ChildNode from the children list of its parent.
ChildNode.replaceWith()
Replaces this ChildNode in the children list of its parent with a set of Node or DOMString objects. DOMString objects are inserted as equivalent Text nodes.

Вставка
ChildNode.before()
Inserts a set of Node or DOMString objects in the children list of this ChildNode's parent, just before this ChildNode. DOMString objects are inserted as equivalent Text nodes.
ChildNode.after()
Inserts a set of Node or DOMString objects in the children list of this ChildNode's parent, just after this ChildNode. DOMString objects are inserted as equivalent Text nodes.

API Element

Большая часть нужных свойств и методов лежит в API Element.
Здесь и навигация, и кол-во наследников, и события мыши, и innerHTML.
Перечислены НЕ все свойства и методы

API Element
API Element => Properties
API Element => Methods

Атрибуты
Element.attributes
Returns a NamedNodeMap object containing the assigned attributes of the corresponding HTML element.
Element.id
Is a DOMString representing the id of the element.

Element.getAttribute()
Retrieves the value of the named attribute from the current node and returns it as an Object.
Element.getAttributeNames()
Returns an array of attribute names from the current element.
Element.hasAttribute()
Returns a Boolean indicating if the element has the specified attribute or not.
Element.hasAttributes()
Returns a Boolean indicating if the element has one or more HTML attributes present.
Element.removeAttribute()
Removes the named attribute from the current node.
Element.setAttribute()
Sets the value of a named attribute of the current node.
Element.setAttributeNS()
Sets the value of the attribute with the specified name and namespace, from the current node.
Element.toggleAttribute()
Toggles a boolean attribute, removing it if it is present and adding it if it is not present, on the specified element.

Выбор элементов на странице
Element.querySelector()
Returns the first Node which matches the specified selector string relative to the element.
Element.querySelectorAll()
Returns a NodeList of nodes which match the specified selector string relative to the element.

Element.getElementsByClassName()
Returns a live HTMLCollection that contains all descendants of the current element that possess the list of classes given in the parameter.
Element.getElementsByTagName()
Returns a live HTMLCollection containing all descendant elements, of a particular tag name, from the current element.

Element.closest()
Returns the Element which is the closest ancestor of the current element (or the current element itself) which matches the selectors given in parameter.

Узнать имя тега
Element.tagName
Returns a String with the name of the tag for the given element.

Проверки
Element.matches()
Returns a Boolean indicating whether or not the element would be selected by the specified selector string.

Дочерние элементы
Element.childElementCount
Returns the number of child elements of this element.

Классы и стили
Element.classList
Returns a DOMTokenList containing the list of class attributes.
Element.className
Is a DOMString representing the class of the element.

Element.computedStyleMap()
Returns a StylePropertyMapReadOnly interface which provides a read-only representation of a CSS declaration block that is an alternative to CSSStyleDeclaration.

Размеры и габариты
Element.clientHeight
Returns a Number representing the inner height of the element.
Element.clientWidth
Returns a Number representing the inner width of the element.

Element.clientTop Read only
Returns a Number representing the width of the top border of the element.
Element.clientLeft
Returns a Number representing the width of the left border of the element.

Element.getBoundingClientRect()
Returns the size of an element and its position relative to the viewport.
Element.getClientRects()
Returns a collection of rectangles that indicate the bounding rectangles for each line of text in a client.

Вставки, содержимое, изменение элементов
Element.innerHTML
Is a DOMString representing the markup of the element's content.
Element.outerHTML
Is a DOMString representing the markup of the element including its content. When used as a setter, replaces the element with nodes parsed from the given string.

Element.insertAdjacentElement()
Inserts a given element node at a given position relative to the element it is invoked upon.
Element.insertAdjacentHTML()
Parses the text as HTML or XML and inserts the resulting nodes into the tree in the position given.
Element.insertAdjacentText()
Inserts a given text node at a given position relative to the element it is invoked upon.
ChildNode.remove()
Removes the element from the children list of its parent.

Навигация
Element.nextElementSibling
Is an Element, the element immediately following the given one in the tree, or null if there's no sibling node.
Element.previousElementSibling
Is a Element, the element immediately preceding the given one in the tree, or null if there is no sibling element.

Прокрутка
Element.scrollHeight
Returns a Number representing the scroll view height of an element.
Element.scrollWidth
Returns a Number representing the scroll view width of the element.

Element.**scrollLeft**
Is a Number representing the left scroll offset of the element.
Element.**scrollTop**
A Number representing number of pixels the top of the document is scrolled vertically.

Element.scrollLeftMax
Returns a Number representing the maximum left scroll offset possible for the element.
Element.scrollTopMax
Returns a Number representing the maximum top scroll offset possible for the element.

Element.scroll()
Scrolls to a particular set of coordinates inside a given element.
Element.scrollBy()
Scrolls an element by the given amount.
Element.scrollIntoView()
Scrolls the page until the element gets into the view.
Element.scrollTo()
Scrolls to a particular set of coordinates inside a given element.

Обработчики событий
EventTarget.addEventListener()
Registers an event handler to a specific event type on the element.
EventTarget.removeEventListener()
Removes an event listener from the element.

Захват элемента
Element.setPointerCapture()
Designates a specific element as the capture target of future pointer events.
Element.releasePointerCapture()
Releases (stops) pointer capture that was previously set for a specific pointer event.

ХЗЧ
Element.assignedSlotRead only
Returns a HTMLSlotElement representing the <slot> the node is inserted in.
Element.localName Read only
A DOMString representing the local part of the qualified name of the element.
Element.namespaceURI Read only
The namespace URI of the element, or null if it is no namespace.
Element.part
Represents the part identifier(s) of the element (i.e. set using the part attribute), returned as a DOMTokenList.
Element.prefix Read only
A DOMString representing the namespace prefix of the element, or null if no prefix is specified.
Element.shadowRoot
Returns the open shadow root that is hosted by the element, or null if no open shadow root is present.
Element.openOrClosedShadowRoot
Returns the shadow root that is hosted by the element, regardless if its open or closed.
Element.slot
Returns the name of the shadow DOM slot the element is inserted in.
Element.tabStop
Is a Boolean indicating if the element can receive input focus via the tab key.
Element.setCapture()
Sets up mouse event capture, redirecting all mouse events to this element.

API Element =>
Events
Listen to these events using addEventListener() or by assigning an event listener to the oneventname property of this interface.

scroll
Fired when the document view or an element has been scrolled.
Also available via the onscroll property.
wheel
Fired when the user rotates a wheel button on a pointing device (typically a mouse).
Also available via the onwheel property.

Clipboard events
copy
Fired when the user initiates a copy action through the browser's user interface.
Also available via the oncopy property.
cut
Fired when the user initiates a cut action through the browser's user interface.
Also available via the oncut property.
paste
Fired when the user initiates a paste action through the browser's user interface.
Also available via the onpaste property.

Focus events
blur
Fired when an element has lost focus.
Also available via the onblur property.
focus
Fired when an element has gained focus.
Also available via the onfocus property
focusin
Fired when an element is about to gain focus.
focusout
Fired when an element is about to lose focus.

Fullscreen events
Keyboard events
keydown
Fired when a key is pressed.
Also available via the onkeydown property.
keypress
Fired when a key that produces a character value is pressed down.
Also available via the onkeypress property.
keyup
Fired when a key is released.
Also available via the onkeyup property.

Mouse events (это именно события)
mousedown
Fired when a pointing device button is pressed on an element.
Also available via the onmousedown property.
mouseup
Fired when a pointing device button is released on an element.
Also available via the onmouseup property.

click
Fired when a pointing device button (e.g., a mouse's primary button) is pressed and released on a single element.
Also available via the onclick property.
dblclick
Fired when a pointing device button (e.g., a mouse's primary button) is clicked twice on a single element.
Also available via the ondblclick property.
contextmenu
Fired when the user attempts to open a context menu.
Also available via the oncontextmenu property.

mousemove
Fired when a pointing device (usually a mouse) is moved while over an element.
Also available via the onmousemove property.

mouseover всплывает
Fired when a pointing device is moved onto the element to which the listener is attached or onto one of its children.
Also available via the onmouseover property.
mouseout

Fired when a pointing device (usually a mouse) is moved off the element to which the listener is attached or off one of its children.
Also available via the onmouseout property.

mouseenter не всплывает
Fired when a pointing device (usually a mouse) is moved over the element that has the listener attached.
Also available via the onmouseenter property.
mouseleave
Fired when the pointer of a pointing device (usually a mouse) is moved out of an element that has the listener attached to it.
Also available via the onmouseleave property.

auxclick
Fired when a non-primary pointing device button (e.g., any mouse button other than the left button) has been pressed and released on an element.
Also available via the onauxclick property.

API HTMLElement

Некоторые свойства:

HTMLElement.accessKey
Is a DOMString representing the access key assigned to the element.

Скрыть элемент
HTMLElement.**hidden**
Is a Boolean indicating if the element is hidden or not.

HTMLOrForeignElement.tabIndex
Is a long representing the position of the element in the tabbing order.

Содержимое
HTMLElement.innerText
Represents the "rendered" text content of a node and its descendants. As a getter, it approximates the text the user would get if they highlighted the contents of the element with the cursor and then copied it to the clipboard.

Габариты
HTMLElement.offsetHeight
Returns a double containing the height of an element, relative to the layout.
HTMLElement.offsetWidth
Returns a double containing the width of an element, relative to the layout.

HTMLElement.offsetTop
Returns a double, the distance from this element's top border to its offsetParent's top border.
HTMLElement.offsetLeft
Returns a double, the distance from this element's left border to its offsetParent's left border.
HTMLElement.offsetParent
Returns a Element that is the element from which all offset calculations are currently computed.

Стили
ElementCSSInlineStyle.style
Is a CSSStyleDeclaration, an object representing the declarations of an element's style attributes.

Методы
HTMLOrForeignElement.blur()
Removes keyboard focus from the currently focused element.
HTMLElement.click()
Sends a mouse click event to the element.
HTMLOrForeignElement.focus()
Makes the element the current keyboard focus.

API HTMLElement =>
Events (некоторые)

Input events
beforeinput
Fired when the value of an <input>, <select>, or <textarea> element is about to be modified.
input
Fired when the value of an <input>, <select>, or <textarea> element has been changed.
Also available via the oninput property.
change
Fired when the value of an <input>, <select>, or <textarea> element has been changed and committed by the user. Unlike the input event, the change event is not necessarily fired for each alteration to an element's value.

Pointer events
Transition events

API Event

Список **свойств** Event тут: **Свойства**
Список **методов** Event тут: **Методы**

Некоторые свойства и методы:

Event()
Создаёт объект Event и возвращает его вызывающему.

Свойства
Event.type
Название события.

Event.target
Ссылка на целевой объект, на котором произошло событие.
Event.currentTarget
Ссылка на текущий зарегистрированный объект, на котором обрабатывается событие. Это объект, которому планируется отправка события; поведение можно изменить с использованием перенаправления (*retargeting*).

Event.bubbles
Логическое значение, указывающее, всплыло ли событие вверх по DOM или нет.
Event.defaultPrevented
Показывает, была ли для события вызвана функция event.preventDefault().

Event.eventPhase
Указывает фазу процесса обработки события.
Event.timeStamp
Время, когда событие было создано (в миллисекундах).

Event.isTrusted
Показывает было или нет событие инициировано браузером (например, по клику мышью) или из скрипта (например, через функцию создания события, такую как event.initEvent)
Event.deepPath
Массив DOM-узлов, через которые всплывало событие.

Методы
Event.preventDefault()
Отменяет действие браузера по умолчанию.
Event.stopImmediatePropagation()
Для конкретного события не будет больше вызвано обработчиков. Дальнейшее всплытие (и погружение) останавливается.
Event.stopPropagation()

```
Остановка распространения события далее по DOM.
```

API MouseEvent

Common events using this interface include click, dblclick, mouseup, mousedown.

```
API MouseEvent => Свойства
кнопки
MouseEvent.altKey
true, если клавиша alt была нажата во время движения мыши.
MouseEvent.shiftKey
true если клавиша shift была нажата, когда произошло событие мыши.
MouseEvent.ctrlKey Только для чтения
true, если клавиша control была нажата во время движения мыши.
MouseEvent.metaKey
true, если клавиша meta была нажата во время движения мыши.

MouseEvent.which
Возвращает код последней нажатой клавиши, когда произошло событие мыши.

MouseEvent.button
Представляет код клавиши, нажатой в то время, когда произошло событие мыши.
MouseEvent.buttons
Отображает, какие клавиши были нажаты во время движения мыши.

Координаты
MouseEvent.clientX
Отображение X координат курсора мыши в локальной системе координат (DOM контент).
MouseEvent.clientY
Отображение Y координат курсора мыши в локальной системе координат (DOM контент).

MouseEvent.pageX
Отображает X координат указателя мыши относительно всего документа.
MouseEvent.pageY
Отображает Y координат указателя мыши относительно всего документа.

MouseEvent.screenX
Отображает X координат указателя мыши в пространстве экрана.
MouseEvent.screenY
Отображает Y координат указателя мыши в пространстве экрана.

MouseEvent.offsetX
Отображает X координат указателя мыши относительно позиции границы отступа целевого узла.
MouseEvent.offsetY
Отображает Y координат указателя мыши относительно позиции границы отступа целевого узла.

MouseEvent.relatedTarget
Второстепенная цель события, если таковая есть.
```

API PointerEvent

События указателя (Pointer events) – это современный единый способ обработки ввода с помощью различных указывающих устройств: мышь, стилус, сенсорный экран и других.
Наследует все свойства MouseEvent + дополнительные свойства здесь.

```
Свойства (некоторые)
PointerEvent.pointerId
A unique identifier for the pointer causing the event.
PointerEvent.pointerType
Indicates the device type that caused the event (mouse, pen, touch, etc.)
PointerEvent.isPrimary
Indicates if the pointer represents the primary pointer of this pointer type.

PointerEvent.width
The width (magnitude on the X axis), in CSS pixels, of the contact geometry of the pointer.
PointerEvent.height
The height (magnitude on the Y axis), in CSS pixels, of the contact geometry of the pointer.
PointerEvent.pressure
The normalized pressure of the pointer input in the range 0 to 1, where 0 and 1 represent the
minimum and maximum pressure the hardware is capable of detecting, respectively.


PointerEvent.tangentialPressure
The normalized tangential pressure of the pointer input (also known as barrel pressure or
cylinder stress) in the range -1 to 1, where 0 is the neutral position of the control.
PointerEvent.tiltX
The plane angle (in degrees, in the range of -90 to 90) between the Y–Z plane and the plane
containing both the pointer (e.g. pen stylus) axis and the Y axis.
PointerEvent.tiltY
The plane angle (in degrees, in the range of -90 to 90) between the X–Z plane and the plane
containing both the pointer (e.g. pen stylus) axis and the X axis.
PointerEvent.twist
The clockwise rotation of the pointer (e.g. pen stylus) around its major axis in degrees, with
a value in the range 0 to 359.


События
Есть у мыши
pointerdown
pointerup
pointermove
pointerover
pointerout
pointerenter
pointerleave
Специфичные
pointercancel
gotpointercapture
lostpointercapture
```

API KeyboardEvent

The event type (keydown, keypress, or keyup) identifies what kind of keyboard activity occurred.

Коды клавиш:
KeyboardEvent: code values
Коды и символы можно посмотреть здесь, на w3.

Некоторые свойства и методы

```
KeyboardEvent.code
Returns a DOMString with the code value of the physical key represented by the event.
KeyboardEvent.key
Returns a DOMString representing the key value of the key represented by the event.
KeyboardEvent.repeat
Returns a Boolean that is true if the key is being held down such that it is automatically
repeating.

Нажаты ли клавиши-модификаторы
KeyboardEvent.altKey
Returns a Boolean that is true if the Alt ( Option or ⌥ on OS X) key was active when the key
event was generated.
KeyboardEvent.ctrlKey
```

Returns a Boolean that is true if the Ctrl key was active when the key event was generated.
KeyboardEvent.shiftKey
Returns a Boolean that is true if the Shift key was active when the key event was generated.
KeyboardEvent.metaKey
Returns a Boolean that is true if the Meta key (on Mac keyboards, the ⌘ Command key; on Windows keyboards, the Windows key (⊞)) was active when the key event was generated.

KeyboardEvent.getModifierState()
Returns a Boolean indicating if a modifier key such as Alt, Shift, Ctrl, or Meta, was pressed when the event was created.

</>

Команды по группам

Alert

```
window.alert()
window.alert(message);

window.prompt()
result = window.prompt(message, default);
cancel = null

window.confirm()
result = window.confirm(message);
true / false
```

Базовые элементы

Базовые теги
Document.head
Returns the <head> element of the current document.
Document.body
Returns the <body> or <frameset> node of the current document.
Document.documentElement
Returns the Element that is a direct child of the document. For HTML documents, this is normally the HTMLHtmlElement object representing the document's <html> element.

Навигация по DOM

Дочерние ноды
Node.childNodes
Returns a live NodeList containing all the children of this node (including elements, text and comments). NodeList being live means that if the children of the Node change, the NodeList object is automatically updated.
ParentNode.children
Returns a live HTMLCollection containing all of the Element objects that are children of this ParentNode, omitting all of its non-element nodes.

Node.firstChild
Returns a Node representing the first direct child node of the node, or null if the node has no child.
ParentNode.firstElementChild
Returns the first node which is both a child of this ParentNode and is also an Element, or null if there is none.

Node.lastChild
Returns a Node representing the last direct child node of the node, or null if the node has no child.
ParentNode.lastElementChild
Returns the last node which is both a child of this ParentNode and is an Element, or null if there is none.

Родительские ноды
Node.parentNode
Returns a Node that is the parent of this node. If there is no such node, like if this node is the top of the tree or if doesn't participate in a tree, this property returns null.
Node.parentElement
Returns an Element that is the parent of this node. If the node has no parent, or if that parent is not an Element, this property returns null.

Соседи
Node.nextSibling
Returns a Node representing the next node in the tree, or null if there isn't such node.
Element.nextElementSibling
Is an Element, the element immediately following the given one in the tree, or null if there's no sibling node.

Node.previousSibling
Returns a Node representing the previous node in the tree, or null if there isn't such node.
Element.previousElementSibling
Is a Element, the element immediately preceding the given one in the tree, or null if there is no sibling element.

В то же время, путь любого элемента для js можно найти в консоли через выбор элемента - правая кнопка мыши - copy - copy js path

Поиск и коллекции

Element.querySelector()
Returns the first Node which matches the specified selector string relative to the element.
Element.querySelectorAll()
Returns a NodeList of nodes which match the specified selector string relative to the element.

Document.getElementById()
returns an Element object representing the element whose id property matches the specified string.

Document.getElementsByName()
returns a NodeList Collection of elements with a given name in the document.
Element.getElementsByClassName()
Returns a live HTMLCollection that contains all descendants of the current element that possess the list of classes given in the parameter.
Element.getElementsByTagName()
Returns a live HTMLCollection containing all descendant elements, of a particular tag name, from the current element.

Element.closest()
Returns the Element which is the closest ancestor of the current element (or the current element itself) which matches the selectors given in parameter.

Document.forms	Returns a list of the <form> elements within the current document.
Document.images	Returns a list of the images in the current document.
Document.links	Returns a list of all the hyperlinks in the document.

Создание, копирование, удаление узлов

Создание

```
Document.createElement()   Creates a new element with the given tag name.
Document.createTextNode()  Creates a text node.
Document.createComment()   Creates a new comment node and returns it.
Document.write()           Нет смысла использовать

Копирование
Node.cloneNode([true])  Clone a Node, and optionally, all of its contents (deep copy, if true).

Удаление и перемещение
ChildNode.remove()
Removes this ChildNode from the children list of its parent.
ChildNode.replaceWith()
Replaces this ChildNode in the children list of its parent with a set of Node or DOMString
objects. DOMString objects are inserted as equivalent Text nodes.
```

Вставка нод, текста и кода

```
Перемещение
ChildNode.replaceWith()
Replaces this ChildNode in the children list of its parent with a set of Node or DOMString
objects. DOMString objects are inserted as equivalent Text nodes.


Вставка узлов и строк (безопасно)
ParentNode.append()
добавляет узлы или строки в конец node
Inserts a set of Node objects or DOMString objects after the last child of the ParentNode.
DOMString objects are inserted as equivalent Text nodes.
ParentNode.prepend()
вставляет узлы или строки в начало node
Inserts a set of Node objects or DOMString objects before the first child of the ParentNode.
DOMString objects are inserted as equivalent Text nodes.

ChildNode.before()
вставляет узлы или строки до node
Inserts a set of Node or DOMString objects in the children list of this ChildNode's parent,
just before this ChildNode. DOMString objects are inserted as equivalent Text nodes.
ChildNode.after()
вставляет узлы или строки после node
Inserts a set of Node or DOMString objects in the children list of this ChildNode's parent,
just after this ChildNode. DOMString objects are inserted as equivalent Text nodes.

Вставка HTML кода
Ещё один способ
Element.insertAdjacentHTML(where, html)        универсальный метод по вставке кода html
Element.insertAdjacentElement(where, element)  вставить готовую ноду
Element.insertAdjacentText(where, text)        вставить текст

Inserts a given element node at a given position relative to the element it is invoked upon.
where
"beforebegin"   вставить html непосредственно перед elem
"afterbegin"    вставить html в начало elem
"beforeend"     вставить html в конец elem
"afterend"      вставить html непосредственно после elem


Устаревшие
Node.appendChild(childNode)
Adds the specified childNode argument as the last child to the current node.
If the argument referenced an existing node on the DOM tree, the node will be detached from its
current position and attached at the new position.
Node.insertBefore()
Inserts a Node before the reference node as a child of a specified parent node.
parentElem.replaceChild(node, oldChild)
заменяет oldChild на node среди дочерних элементов parentElem
parentElem.removeChild(node)
удаляет node из parentElem (предполагается, что он родитель node).
```

Содержимое нод: текст и код

```
Element.innerHTML
Содержимое внутри элемента в виде строки. Можно менять.
Is a DOMString representing the markup of the element's content.
Element.outerHTML
Перезапись HTML элемента целиком. Старый элемент не изменяется, а заменяется целиком.
Is a DOMString representing the markup of the element including its content. When used as a
setter, replaces the element with nodes parsed from the given string.

Node.textContent
текст внутри узла-элемента (вычет всех тегов). Можно менять.
Returns / Sets the textual content of an element and all its descendants
HTMLElement.innerText
текстовое содержимое элемента и его потомков.
Represents the "rendered" text content of a node and its descendants. As a getter, it
approximates the text the user would get if they highlighted the contents of the element with
the cursor and then copied it to the clipboard.

Node.nodeValue
Текст внутри ноды.
Returns / Sets the value of the current node.
HTMLObjectElement.data
Текст внутри элемента.
returns a DOMString that reflects the data HTML attribute, specifying the address of a
resource's data.
См. «Как вытащить текст»
```

Свойства нод и проверки

```
Node.nodeName
Returns a DOMString containing the name of the Node.
Element.tagName
Returns a String with the name of the tag for the given element.


Node.nodeType
Старомодный способ узнать тип узла. Returns an unsigned short representing the type of the
node. Possible values here.

Новые методы узнать имя класса DOM-узла
document.body.constructor.name    // HTMLBodyElement
document.body.toString;           // [object HTMLBodyElement]

Event.type
Название события.


Node.contains()
Returns a Boolean value indicating whether or not a node is a descendant of the calling node.
Node.hasChildNodes()
Returns a Boolean indicating whether or not the element has any child nodes.


Element.matches()
Returns a Boolean indicating whether or not the element would be selected by the specified
selector string.

Element.hasAttribute()
```

Returns a Boolean indicating if the element has the specified attribute or not.
Element.hasAttributes()
Returns a Boolean indicating if the element has one or more HTML attributes present.

Document.childElementCount
Returns the number of child elements of the current document
ParentNode.childElementCount
Returns the number of children of this ParentNode which are elements.
Element.childElementCount
Returns the number of child elements of this element.

Класс узла
Можно узнать API таким образом
```
document.body.constructor.name  HTMLBodyElement
document.body.toString;         [object HTMLBodyElement]
```

Скрыть элемент

HTMLElement.**hidden**
Is a Boolean indicating if the element is hidden or not.

Атрибуты

Document.createAttribute()
Creates a new Attr object and returns it.

Element.attributes
Returns a NamedNodeMap object containing the assigned attributes of the corresponding HTML element.
Element.id
Is a DOMString representing the id of the element.

Element.toggleAttribute()
Toggles a boolean attribute, removing it if it is present and adding it if it is not present, on the specified element.

Когда браузер парсит HTML, чтобы создать DOM-объекты для тегов, он распознаёт стандартные атрибуты и создаёт DOM-свойства для них. Браузер НЕ распознаёт нестандартные атрибуты. Нижеследующие методы работают именно с тем, что написано в HTML, поэтому они могут читать нестандартные атрибуты.

Получить доступ и к стандартным, и к пользовательским атрибутам можно с помощью этих методов:
Element.getAttribute()
Retrieves the value of the named attribute from the current node and returns it as an Object.
Element.hasAttribute()
Returns a Boolean indicating if the element has the specified attribute or not.
Element.setAttribute()
Sets the value of a named attribute of the current node.
Element.removeAttribute()
Removes the named attribute from the current node.

Element.getAttributeNames()
Returns an array of attribute names from the current element.
Element.hasAttributes()
Returns a Boolean indicating if the element has one or more HTML attributes present.
Element.setAttributeNS()
Sets the value of the attribute with the specified name and namespace, from the current node.

Dataset

Чтобы избежать конфликтов нестандартных и стандартных атрибутов, все нестандартные атрибуты надо записывать в html с префикса «data-». Такие атрибуты автоматом распарсятся и станут доступны в свойстве DOM-объекта .dataset.
Атрибуты, состоящие из нескольких слов, data-order-state, становятся свойствами, записанными с помощью верблюжьей нотации: .dataset.orderState.

data-*
Определяет группу атрибутов, называемых атрибутами пользовательских данных, позволяющих осуществлять обмен служебной информацией между HTML и его DOM представлением.

HTMLOrForeignElement.dataset
Предоставляет доступ как для чтения, так и для изменения пользовательских data-атрибутов (data-*). Это map of DOMString, одна запись для каждого пользовательского атрибута данных.

Само свойство dataset доступно только для чтения. Для записи должны использоваться его "свойства", которые представлены data-атрибутами.

```
<body data-about="Elephants">
alert(document.body.dataset.about); // Elephants
```

Классы и стили

Классы
Element.className
Получить или записать атрибут элемента class="…" в виде строки
Is a DOMString representing the class of the element.

Element.classList
специальный объект для добавления/удаления класса
Returns a DOMTokenList containing the list of class attributes.

DOMTokenList
```
add('class1', 'class2')     Добавляет элементу указанные классы
remove('class1', 'class2')  Удаляет у элемента указанные классы
contains('class1')          проверка, есть ли данный класс у элемента
item(Number)                аналог вызова classList[index]
length                      кол-во классов у элемента
toggle('class1' [,Boolean]) добавляет или убирает класс. False/true всегда удалить/добавить
```

Стили
ElementCSSInlineStyle.style.property.value =
Is a CSSStyleDeclaration, an object representing the declarations of an element's style attributes.

elem.style.cssText
переписать или получить весь атрибут style

window.getComputedStyle(element)
Возвращает живой объект, содержащий значения всех CSS-свойств элемента, полученных после применения всех активных таблиц стилей, и завершения базовых вычислений.
Возвращает style живой CSSStyleDeclaration объект.

Element.computedStyleMap()
Returns a StylePropertyMapReadOnly interface which provides a read-only representation of a CSS declaration block that is an alternative to CSSStyleDeclaration.

DOMTokenList

It is indexed beginning with 0 as with JavaScript Array objects. DOMTokenList is always case-sensitive.

The **DOMTokenList** interface represents a set of space-separated tokens. Such a set is returned by:
Element.classList
HTMLLinkElement.relList
HTMLAnchorElement.relList
HTMLAreaElement.relList
HTMLIframeElement.sandbox
HTMLOutputElement.htmlFor.

DOMTokenList.length
Is an integer representing the number of objects stored in the object.
DOMTokenList.value
A stringifier property that returns the value of the list as a DOMString.

DOMTokenList.item(*index*)
Returns the item in the list by its *index*, or undefined if *index* is greater than or equal to the list's length.
DOMTokenList.contains(*token*)
Returns true if the list contains the given *token*, otherwise false.

DOMTokenList.add(*token1*[, *token2*[, ...*tokenN*]])
Adds the specified *token*(s) to the list.
DOMTokenList.remove(*token1*[, *token2*[, ...*tokenN*]])
Removes the specified *token*(s) from the list.
DOMTokenList.replace(*oldToken*, *newToken*)
Replaces *token* with *newToken*.
DOMTokenList.toggle(*token* [, *force*])
Removes *token* from the list if it exists, or adds *token* to the list if it doesn't. Returns a boolean indicating whether *token* is in the list after the operation.

DOMTokenList.keys()
Returns an iterator, allowing you to go through all keys of the key/value pairs contained in this object.
DOMTokenList.values()
Returns an iterator, allowing you to go through all values of the key/value pairs contained in this object.
DOMTokenList.entries()
Returns an iterator, allowing you to go through all key/value pairs contained in this object.

DOMTokenList.forEach(*callback* [, *thisArg*])
Executes a provided *callback* function once per DOMTokenList element.

DOMTokenList.supports(*token*)
Returns true if a given *token* is in the associated attribute's supported tokens.

Размеры и габариты

родительский элемент
HTMLElement.offsetParent
ссылка на предка элемента
Returns a Element that is the element from which all offset calculations are currently computed.
HTMLElement.offsetTop
расстояние вверх до родителя
Returns a double, the distance from this element's top border to its offsetParent's top border.
HTMLElement.offsetLeft
расстояние слева от родителя
Returns a double, the distance from this element's left border to its offsetParent's left border.

рамка border
Element.clientTop
Толщина рамки сверху
Returns a Number representing the width of the top border of the element.
Element.clientLeft
Толщина рамки слева
Returns a Number representing the width of the left border of the element.

ширина + paddind
Element.clientHeight
Returns a Number representing the inner height of the element.
Element.clientWidth
Returns a Number representing the inner width of the element.

ширина + padding + border
HTMLElement.offsetHeight
Returns a double containing the height of an element, relative to the layout.
HTMLElement.offsetWidth
Returns a double containing the width of an element, relative to the layout.

Размер элемента с невидимой частью
Это не прокрутка!
Element.scrollHeight
измерение высоты контента в элементе, включая содержимое, невидимое из-за прокрутки.
Element.scrollWidth
измерение ширины контента в элементе, включая содержимое, невидимое из-за прокрутки.

ClientRect
Element.getBoundingClientRect()
Returns the size of an element and its position relative to the viewport.
Element.getClientRects()
Returns a collection of rectangles that indicate the bounding rectangles for each line of text in a client.

Окно
с полосой прокрутки
Window.innerHeight
Gets the height of the content area of the browser window including, if rendered, the horizontal scrollbar.
Window.innerWidth
Gets the width of the content area of the browser window including, if rendered, the vertical scrollbar.

без полосы прокрутки
document.documentElement.clientHeight
document.documentElement.clientWidth

Element.**scrollLeft**
ширина невидимой, уже прокрученной части элемента (можно записывать)
Is a Number representing the left scroll offset of the element.
Element.**scrollTop**
ширина невидимой, уже прокрученной части элемента (можно записывать)
A Number representing number of pixels the top of the document is scrolled vertically.

узнать размер прокрутки снизу
```
let scrollBottom = elem.scrollHeight - elem.scrollTop - elem.clientHeight;
```

Высота всего documentElement
```
let height = Math.max(
  document.body.scrollHeight, document.documentElement.scrollHeight,
```

```
  document.body.offsetHeight, document.documentElement.offsetHeight,
  document.body.clientHeight, document.documentElement.clientHeight
);
```

Прокрутка

```
Window.scroll()
Scrolls the window to a particular place in the document.
Window.scrollBy()
Scrolls the document in the window by the given amount.
Window.scrollTo()
Scrolls to a particular set of coordinates in the document.

ScrollToOptions
Это API для использования вместе с методами .scroll, .scrollBy и .scrollTo.

Window.pageXOffset Read only
An alias for window.scrollX.
Window.pageYOffset Read only
An alias for window.scrollY



Это не прокрутка!
Element.scrollHeight
измерение высоты контента в элементе, включая содержимое, невидимое из-за прокрутки.
Element.scrollWidth
измерение ширины контента в элементе, включая содержимое, невидимое из-за прокрутки.


получить или задать прокрутку элемента (не окна!)
Element.scrollLeft
Is a Number representing the left scroll offset of the element.
Element.scrollTop
A Number representing number of pixels the top of the document is scrolled vertically.


Element.scrollLeftMax
Returns a Number representing the maximum left scroll offset possible for the element.
Element.scrollTopMax
Returns a Number representing the maximum top scroll offset possible for the element.

Element.scroll()
Scrolls to a particular set of coordinates inside a given element.
Element.scrollBy()
Scrolls an element by the given amount.
Element.scrollIntoView()
Scrolls the page until the element gets into the view.
Element.scrollTo()
Scrolls to a particular set of coordinates inside a given element.


document.body.style.overflow = "hidden"   отключить прокрутку
document.body.style.overflow = ""         включить обратно

узнать размер прокрутки снизу (сколько ещё крутить осталось)
let scrollBottom = elem.scrollHeight - elem.scrollTop - elem.clientHeight;

Высота всего documentElement
let height = Math.max(
  document.body.scrollHeight, document.documentElement.scrollHeight,
  document.body.offsetHeight, document.documentElement.offsetHeight,
  document.body.clientHeight, document.documentElement.clientHeight
);
```

Элементы из точки

```
Document.elementFromPoint()
Returns the topmost element at the specified coordinates.
Document.elementsFromPoint()
Returns an array of all elements at the specified coordinates.
```

Таблицы

Некоторые типы DOM-элементов предоставляют для удобства дополнительные свойства, специфичные для их типа. API по таблицам:
HTMLTableCaptionElement
HTMLTableCellElement
HTMLTableColElement
HTMLTableElement
HTMLTableRowElement
HTMLTableSectionElement

```
<table> дополнительно имеет
HTMLTableElement
Свойства
table.caption  возвращает заголовок таблицы.
table.tHead    возвращает header таблицы.
table.tFoot    возвращает footer таблицы.
table.rows     возвращает строки таблицы.
table.tBodies  возвращает тела таблицы.

table.align        возвращает/устанавливает выравнивание таблицы.
table.bgColor      возвращает/устанавливает цвет фона таблицы.
table.border       возвращает/устанавливает рамку таблицы.
table.cellPadding  возвращает/устанавливает cellpadding.
table.cellSpacing  возвращает/устанавливает cellspacing.
table.frame        определяет, какие стороны таблицы имеют рамку.
table.rules        определяет, какие внутренние рамки являются видимыми.
table.summary      возвращает/устанавливает описание таблицы.
table.width        возвращает/устанавливает ширину таблицы.

Методы()
table.createTHead    создает header таблицы.
table.deleteTHead    удаляет header таблицы.
table.createTFoot    создает footer таблицы.
table.deleteTFoot    удаляет footer таблицы.
table.insertRow      вставляет строку.
table.deleteRow      удаляет строку.
table.createCaption  создает заголовок таблицы.
table.deleteCaption  удаляет заголовок таблицы.


<tr>
HTMLTableRowElement
tr.cells             живая коллеция ячеек в ряду
tr.rowIndex          номер строки <tr> в таблице (включая все строки таблицы)
tr.sectionRowIndex   номер строки <tr> в текущей секции <thead>/<tbody>/<tfoot>.

Методы
tr.deleteCell()      удалить ячейку
tr.insertCell()      вставить ячейку

<thead>, <tfoot>, <tbody>
```

```
.rows

<td> <th>
.cellIndex      номер ячейки в строке <tr>

// выбрать первую таблицу в документе
a = document.getElementsByTagName('table')[0]

b = a.rows
// записать в переменную все tr, что есть в таблице

c = a.rows.cells[0]
// записать первую ячеку
```

Формы

```
Document.forms
Returns a list of the <form> elements within the current document.

index access
let selectForm = document.forms[index];
let selectFormElement = document.forms[index].elements[index];

named form access
let selectForm = document.forms.login;
let selectForm = document.forms['login'];

loginForm.elements.email.placeholder = 'test@example.com';
loginForm.elements.password.placeholder = 'password';

элементы формы ссылаются на форму
formElement.form === form


Отправка формы
HTMLFormElement: submit event
fires when a <form> is submitted.

HTMLFormElement.submit()
method submits a given <form>
This method is similar, but not identical to, activating a form's submit <button>. When
invoking this method directly, however:
- No submit event is raised. In particular, the form's onsubmit event handler is not run.
- Constraint validation is not triggered.

HTMLFormElement.requestSubmit()
requests that the form be submitted using a specific submit button.


Элементы формы
HTMLInputElement
form
value  текст внутри

Properties that apply only to elements of type checkbox or radio
checked
defaultChecked

HTMLTextAreaElement
form
value       текст внутри
autofocus
rows
cols
Methods
blur()
focus()

HTMLSelectElement
.autofocus
.form
.labels
.options          коллекция из подэлементов <option>
.value            значение выбранного в данный момент <option>
.selectedIndex    индекс выбранного <option>
.multiple         возможность выбрать несколько <option>
.selectedOptions
Methods
.add()
.remove()
.item()

HTMLOptionElement
.form
.selected         выбрана ли опция
.defaultSelected  будет ли опция выбрана по умолчанию
.index            номер опции среди других в списке <select>
.label
.text             текст, который видит посетитель
.value            значение, которое отправится на сервер

new Option()      конструктор для создания <option>
option = new Option(text, value, defaultSelected, selected);
text              текст внутри <option>,
value             значение
defaultSelected   если true, то ставится HTML-атрибут selected,
selected          если true, то элемент <option> будет выбранным.
```

Фокус, блюр

```
события
не всплывают
focus    событие вызывается в момент фокусировки
blur     событие вызывается в момент потери фокуса
всплывают
focusin  fired when an element is about to gain focus
focusout fired when an element is about to lose focus


методы
HTMLOrForeignElement.focus() Makes the element the current keyboard focus.
HTMLOrForeignElement.blur()  Removes keyboard focus from the currently focused element.

Document.activeElement       Returns the Element that currently has focus.
HTMLSelectElement.autofocus  whether the control should have input focus when the page loads.

HTMLOrForeignElement.tabIndex
Is a long representing the position of the element in the tabbing order.
Значения
положительное  порядковый номер переключения через Tab
отрицательное  поддерживает фокус, но не выделяется с помощью Tab
0              элемент не поддерживает фокус
```

Event, объект

Event()
Создаёт объект Event и возвращает его вызывающему.

Свойства
Event.type
Название события.

Event.target
Ссылка на целевой объект, на котором произошло событие.
Event.currentTarget
Ссылка на текущий зарегистрированный объект, на котором обрабатывается событие. Это объект, которому планируется отправка события; поведение можно изменить с использованием перенаправления (*retargeting*).

Event.bubbles
Логическое значение, указывающее, всплыло ли событие вверх по DOM или нет.
Event.defaultPrevented
Показывает, была ли для события вызвана функция event.preventDefault().

Event.eventPhase
Указывает фазу процесса обработки события.
Event.timeStamp
Время, когда событие было создано (в миллисекундах).

Event.isTrusted
Показывает было или нет событие инициировано браузером (например, по клику мышью) или из скрипта (например, через функцию создания события, такую как event.initEvent)
Event.deepPath
Массив DOM-узлов, через которые всплывало событие.

Методы
Event.preventDefault()
Отменяет действие браузера по умолчанию.

Event.stopPropagation()
Остановка распространения события далее по DOM.
Event.stopImmediatePropagation()
Для конкретного события не будет больше вызвано обработчиков. Дальнейшее всплытие (и погружение) останавливается.

MouseEvent, объект

API MouseEvent => Свойства
кнопки
MouseEvent.altKey
true, если клавиша alt была нажата во время движения мыши.
MouseEvent.shiftKey
true если клавиша shift была нажата, когда произошло событие мыши.
MouseEvent.ctrlKey Только для чтения
true, если клавиша control была нажата во время движения мыши.
MouseEvent.metaKey
true, если клавиша meta была нажата во время движения мыши.

MouseEvent.which
Возвращает код последней нажатой клавиши, когда произошло событие мыши.

MouseEvent.button
Представляет код клавиши, нажатой в то время, когда произошло событие мыши.
MouseEvent.buttons
Отображает, какие клавиши были нажаты во время движения мыши.

Координаты
MouseEvent.clientX
Отображение X координат курсора мыши в локальной системе координат (DOM контент).
MouseEvent.clientY
Отображение Y координат курсора мыши в локальной системе координат (DOM контент).

MouseEvent.pageX
Отображает X координат указателя мыши относительно всего документа.
MouseEvent.pageY
Отображает Y координат указателя мыши относительно всего документа.

MouseEvent.screenX
Отображает X координат указателя мыши в пространстве экрана.
MouseEvent.screenY
Отображает Y координат указателя мыши в пространстве экрана.

MouseEvent.offsetX
Отображает X координат указателя мыши относительно позиции границы отступа целевого узла.
MouseEvent.offsetY
Отображает Y координат указателя мыши относительно позиции границы отступа целевого узла.

MouseEvent.relatedTarget
Второстепенная цель события, если таковая есть.

PointerEvent, объект

Element.setPointerCapture()
Designates a specific element as the capture target of future pointer events.
Element.releasePointerCapture()
Releases (stops) pointer capture that was previously set for a specific pointer event.

PointerEvent.pointerId
A unique identifier for the pointer causing the event.
PointerEvent.pointerType
Indicates the device type that caused the event (mouse, pen, touch, etc.)
PointerEvent.isPrimary
Indicates if the pointer represents the primary pointer of this pointer type.

PointerEvent.width
The width (magnitude on the X axis), in CSS pixels, of the contact geometry of the pointer.
PointerEvent.height
The height (magnitude on the Y axis), in CSS pixels, of the contact geometry of the pointer.
PointerEvent.pressure
The normalized pressure of the pointer input in the range 0 to 1, where 0 and 1 represent the minimum and maximum pressure the hardware is capable of detecting, respectively.

PointerEvent.tangentialPressure
The normalized tangential pressure of the pointer input (also known as barrel pressure or cylinder stress) in the range -1 to 1, where 0 is the neutral position of the control.
PointerEvent.tiltX
The plane angle (in degrees, in the range of -90 to 90) between the Y–Z plane and the plane containing both the pointer (e.g. pen stylus) axis and the Y axis.
PointerEvent.tiltY
The plane angle (in degrees, in the range of -90 to 90) between the X–Z plane and the plane containing both the pointer (e.g. pen stylus) axis and the X axis.
PointerEvent.twist
The clockwise rotation of the pointer (e.g. pen stylus) around its major axis in degrees, with a value in the range 0 to 359.

KeyboardEvent, объект

The event type (keydown, keypress, or keyup) identifies what kind of keyboard activity occurred.
KeyboardEvent: code values
Коды и символы можно посмотреть здесь, на w3.

```
KeyboardEvent.code
Returns a DOMString with the code value of the physical key represented by the event.
KeyboardEvent.key
Returns a DOMString representing the key value of the key represented by the event.
KeyboardEvent.repeat
Returns a Boolean that is true if the key is being held down such that it is automatically
repeating.

Клавиши-модификаторы
KeyboardEvent.altKey
Returns a Boolean that is true if the Alt ( Option or ⌥ on OS X) key was active when the key
event was generated.
KeyboardEvent.ctrlKey
Returns a Boolean that is true if the Ctrl key was active when the key event was generated.
KeyboardEvent.shiftKey
Returns a Boolean that is true if the Shift key was active when the key event was generated.
KeyboardEvent.metaKey
Returns a Boolean that is true if the Meta key (on Mac keyboards, the ⌘ Command key; on
Windows keyboards, the Windows key (⊞)) was active when the key event was generated.

KeyboardEvent.getModifierState()
Returns a Boolean indicating if a modifier key such as Alt, Shift, Ctrl, or Meta, was pressed
when the event was created.
```

События, перечень

Перечень событий в DOM
https://developer.mozilla.org/ru/docs/Web/Events

```
API Element => Events
Listen to these events using addEventListener() or by assigning an event listener to the
oneventname property of this interface.

scroll
Fired when the document view or an element has been scrolled.
Also available via the onscroll property.
wheel
Fired when the user rotates a wheel button on a pointing device (typically a mouse).
Also available via the onwheel property.

Focus events
blur
Fired when an element has lost focus.
Also available via the onblur property.
focus
Fired when an element has gained focus.
Also available via the onfocus property
focusin
Fired when an element is about to gain focus.
focusout
Fired when an element is about to lose focus.

Fullscreen events
Keyboard events
keydown
Fired when a key is pressed.
Also available via the onkeydown property.
keypress
Fired when a key that produces a character value is pressed down.
Also available via the onkeypress property.
keyup
Fired when a key is released.
Also available via the onkeyup property.

Mouse events (это именно события)
mousedown
Fired when a pointing device button is pressed on an element.
Also available via the onmousedown property.
mouseup
Fired when a pointing device button is released on an element.
Also available via the onmouseup property.

click
Fired when a pointing device button (e.g., a mouse's primary button) is pressed and released on
a single element.
Also available via the onclick property.
dblclick
Fired when a pointing device button (e.g., a mouse's primary button) is clicked twice on a
single element.
Also available via the ondblclick property.
contextmenu
Fired when the user attempts to open a context menu.
Also available via the oncontextmenu property.

mousemove
Fired when a pointing device (usually a mouse) is moved while over an element.
Also available via the onmousemove property.

mouseover всплывает
Fired when a pointing device is moved onto the element to which the listener is attached or
onto one of its children.
Also available via the onmouseover property.
mouseout
Fired when a pointing device (usually a mouse) is moved off the element to which the listener
is attached or off one of its children.
Also available via the onmouseout property.

mouseenter не всплывает
Fired when a pointing device (usually a mouse) is moved over the element that has the listener
attached.
Also available via the onmouseenter property.
mouseleave
Fired when the pointer of a pointing device (usually a mouse) is moved out of an element that
has the listener attached to it.
Also available via the onmouseleave property.

auxclick
Fired when a non-primary pointing device button (e.g., any mouse button other than the left
button) has been pressed and released on an element.
Also available via the onauxclick property.


Ввод и изменение данных
API HTMLElement => Events (некоторые)
Input events
beforeinput
Fired when the value of an <input>, <select>, or <textarea> element is about to be modified.
input
Fired when the value of an <input>, <select>, or <textarea> element has been changed.
Also available via the oninput property.
change
Fired when the value of an <input>, <select>, or <textarea> element has been changed and
committed by the user. Unlike the input event, the change event is not necessarily fired for
each alteration to an element's value.
```

Clipboard events (Window API**)**
copy
Fired when the user initiates a copy action through the browser's user interface.
Also available via the oncopy property.
cut
Fired when the user initiates a cut action through the browser's user interface.
Also available via the oncut property.
paste
Fired when the user initiates a paste action through the browser's user interface.
Also available via the onpaste property.

Pointer events
Есть у мыши
pointerdown
pointerup
pointermove
pointerover
pointerout
pointerenter
pointerleave
Специфичные
pointercancel
gotpointercapture
lostpointercapture

Transition events

API Form
HTMLFormElement: submit event
fires when a <form> is submitted.

API Window events
Load & unload events
Window: DOMContentLoaded
Document: DOMContentLoaded event
Событие DOMContentLoaded запускается когда первоначальный HTML документ будет полностью загружен и разобран, без ожидания полной загрузки таблиц стилей, изображений и фреймов. См. исключения в конспекте.
Window: load event
Событие происходит когда ресурс и его зависимые ресурсы закончили загружаться.
Window: beforeunload event
Событие запускается, когда окно, документ и его ресурсы вот-вот будут выгружены. Документ все ещё виден, и событие в этот момент может быть отменено.
Window: unload event
The unload event is fired when the document or a child resource is being unloaded.

Document: readystatechange event
The readystatechange event is fired when the readyState attribute of a document has changed.

Обработчики событий

EventTarget.addEventListener('event', handler [, options])
Registers an event handler to a specific event type on the element.
EventTarget.removeEventListener('event', handler [, options])
Removes an event listener from the element.

```
Options {           объект со свойствами
  once: true,       обработчик автоматом удалится после выполнения
  capture: false,   фаза, на которой сработает обработчик. Если true – сработает на погружении.
  passive: true     обработчик никогда не вызовет preventDefault()
}
```

Ввод, изменение, копирование данных

API HTMLElement => Events
Input events
input
Fired when the value of an <input>, <select>, or <textarea> element has been changed.
Also available via the oninput property.
change
Fired when the value of an <input>, <select>, or <textarea> element has been changed and committed by the user. Unlike the input event, the change event is not necessarily fired for each alteration to an element's value.

Clipboard events
copy
Fired when the user initiates a copy action through the browser's user interface.
Also available via the oncopy property.
cut
Fired when the user initiates a cut action through the browser's user interface.
Also available via the oncut property.
paste
Fired when the user initiates a paste action through the browser's user interface.
Also available via the onpaste property.

API ClipboardEvent
The **ClipboardEvent** interface represents events providing information related to modification of the clipboard, that is cut, copy, and paste events.

ClipboardEvent()
Creates a ClipboardEvent event with the given parameters.
ClipboardEvent.clipboardData Read only
Is a DataTransfer object containing the data affected by the user-initiated cut, copy, or paste operation, along with its MIME type.

API DataTransfer
только 3 метода из множества
DataTransfer.clearData()
Remove the data associated with a given type. The type argument is optional. If the type is empty or not specified, the data associated with all types is removed. If data for the specified type does not exist, or the data transfer contains no data, this method will have no effect.
DataTransfer.getData(format)
Retrieves the data for a given type, or an empty string if data for that type does not exist or the data transfer contains no data.
Format - Example data types are «text/plain» and «text/uri-list».
DataTransfer.setData()
Set the data for a given type. If data for the type does not exist, it is added at the end, such that the last item in the types list will be the new format. If data for the type already exists, the existing data is replaced in the same position.

Загрузка документа и ресурсов

Загрузка страницы
События
Document: DOMContentLoaded event
Событие DOMContentLoaded запускается когда первоначальный HTML документ будет полностью загружен и разобран, без ожидания полной загрузки таблиц стилей, изображений и фреймов. См. исключения в конспекте.

```
Window: load event
Событие происходит когда ресурс и его зависимые ресурсы закончили загружаться.

Window: beforeunload event
Событие запускается, когда окно, документ и его ресурсы вот-вот будут выгружены. Документ все ещё виден, и событие в этот момент может быть отменено.

Window: unload event
The unload event is fired when the document or a child resource is being unloaded.


Document.readyState
Свойство описывает состояние загрузки document. Состояния: loading interactive complete. Может быть альтернативой событию load.

Document: readystatechange event
The readystatechange event is fired when the readyState attribute of a document has changed.

Navigator.sendBeacon()
Используется для асинхронной передачи информации до 64кб, в основном аналитики, веб-серверу. Используется в событии unload.


Загрузка ресурсов

onload
onerror


Выполнение скриптов
HTMLScriptElement
HTMLScriptElement.defer
- загружается в фоновом режиме, браузер продолжит обрабатывать страницу;
- выполнится только после готовности DOM-дерева, но до события DOMContentLoaded;
- несколько скриптов загружаются параллельно, но выполняются последовательно, как расположены в документе;
- работает только со внешними скриптами.

HTMLScriptElement.async
- загружается в фоновом режиме;
- выполнится сразу же, не ждёт DOMContentLoaded;
- несколько скриптов загружаются параллельно и выполняются в порядке загрузки (не ждут друг друга).
```

</>

Команды (старая запись)

Поиск в DOM

Это получение произвольных элементов без перемещения по дереву.

Все методы getElementsBy* возвращают живую коллекцию HTMLCollection. Такие коллекции всегда отражают текущее состояние документа и автоматически обновляются при его изменении. Важно, что они возвращают именно коллекцию, а не элемент, даже если элемент в ней один.

querySelectorAll возвращает статическую коллекцию.

Путь любого элемента для js можно найти в консоли через выбор элемента - правая кнопка мыши - copy - copy js path

```
document.getElementById('id')

elem.querySelectorAll(css)
elem.querySelector(css)

elem.getElementsByClassName(className)
document.getElementsByName(name)
elem.getElementsByTagName(tag)    * вместо тега вернёт всех потомков

elem.closest(css)     ближайший соответствующий правилу предок
elem.matches(css)     проверяет, удовлетворяет ли elem CSS-селектору
lemA.contains(elemB)  проверяет на наличие потомка внутри
```

document.getElementById

Если у элемента есть атрибут id, то мы можем получить его вызовом document.getElementById(id).
Метод можно вызвать только на объекте document, потому что он осуществляет поиск всего одного элемента по всему документу сразу.
Метод ищет по DOM.

```
let elem = document.getElementById('id');
```

.querySelectorAll(css)

Самый универсальный поиск. Возвращает все элементы внутри элемента, на котором применяется, удовлетворяющие CSS-селектору. Возвращает статическую коллекцию NodeList. Это похоже на фиксированный массив элементов.
Метод ищет по DOM.

Можно использовать любой CSS-селектор, в т.ч. псевдоклассы :hover, :active и другие.
Например, document.querySelectorAll(':hover') вернёт коллекцию (в порядке вложенности: от внешнего к внутреннему) из текущих элементов под курсором мыши.

Получить все элементы `<li>`, которые являются последними потомками в `<ul>`:

```
let elements = document.querySelectorAll('ul > li:last-child');
```

.querySelector(css)

Возвращает первый элемент, соответствующий переданному CSS-селектору. Метод ищет по DOM.

.querySelectorAll[0] сначала найдёт все элементы по селектору, а потом вернёт первый.
.querySelector найдёт только первый и остановится. Это быстрее.

В формате строки можно передать такую конструкцию:

```
document.querySelector('form[name="search"]');
```

Таким образом будет найден первый элемент «form» с именем name="search".

.getElementsByTagName(tag)

ищет элементы с данным тегом и возвращает их коллекцию.
Передав "*" вместо тега, можно получить всех потомков.

```
let divs = document.getElementsByTagName('div');
```

.getElementsByClassName(className)

Возвращает элементы, которые имеют данный CSS-класс.

```
let articles = form.getElementsByClassName('article');
```

document.getElementsByName(name)

Возвращает элементы с заданным атрибутом name. Очень редко используется.

```
let form = document.getElementsByName('my-form');
```

Проверка соответствия

Систематизация

```
Проверка соответствия
elem.closest(css)       ближайший соответствующий предок
elem.matches(css)       проверяет, удовлетворяет ли elem CSS-селектору
lemA.contains(elemB)    проверяет на наличие потомка внутри
node.hasChildNodes()
node.childElementCount
```

.closest(css)

Метод elem.closest(css) ищет ближайшего предка (по цепочке дерева вверх), который соответствует CSS-селектору. Сам элемент также включается в поиск.
Метод поднимается вверх от элемента, на котором применяется, и проверяет каждого из родителей. Если он соответствует селектору, поиск прекращается.
Возвращает либо предка, либо null, если такой элемент не найден.

```
<h1>Содержание</h1>

<div class="contents">
  <ul class="book">
    <li class="chapter">Глава 1</li>
    <li class="chapter">Глава 2</li>
  </ul>
</div>

<script>
  let chapter = document.querySelector('.chapter'); // LI

  chapter.closest('.book');     // UL
  chapter.closest('.contents'); // DIV

  chapter.closest('h1');        // null (h1 - не предок)
</script>
```

.matches(css)

Проверка соответствия элемента указанному css-селектору.
Вернёт true или false. MDN
Удобен при переборе коллекции и выборке нужных элементов:

```
for (let elem of document.body.children) {
    if (elem.matches('a[href$="zip"]')) {
      alert("Ссылка на архив: " + elem.href );
    }
  }
```

node.hasChildNodes()

Возвращает true или false, есть ли у элемента дочерние узлы или нет.

```
node.hasChildNodes()
проверить наличие дочерних узлов
```

node.contains

возвращает Boolean значение, указывающее, является ли узел потомком данного узла (все уровни вложенности):

```
node.contains( otherNode )
```

node.childElementCount

Возвращает число дочерних **элементов** узла.
Можно применять к Document, DocumentFragment или Element

```
var elCount = Node.childElementCount;
```

Этого свойства нет в учебнике.

Получение узлов (навигация)

Систематизация

DOM-коллекции и все навигационные свойства доступны только для чтения.

```
Верхушка
<html> = document.documentElement
<head> = document.head
<body> = document.body

node
.childNodes
.firstChild
.lastChild
.hasChildNodes() функция проверки дочерних узлов

.nextSibling
.previousSibling.

.parentNode

element
.children
.firstElementChild
.lastElementChild

.nextElementSibling
.previousElementSibling.

.parentElement
```

```
document.body.childNodes
обратиться к дочерним элементам

element.firstChild
element.lastChild
Обратиться к первому и последнему дочернему элементу

elem.firstChild === elem.childNodes[0]
elem.lastChild  === elem.childNodes[elem.childNodes.length - 1]
```

Навигация по узлам

Узлы – это и теги, и текст, и комментарии, и всё остальное.
API - Node

node.childNodes

Коллекция детей элемента, включая текстовые узлы.
Это не массив, а коллекция – особый перебираемый объект-псевдомассив. С ним можно использовать for…of, или перегнать в массив:

```
Array.from(document.body.childNodes);
```

node.firstChild

node.lastChild

Обратиться к первому и последнему дочернему узлу соответственно.
Они, по сути, являются всего лишь сокращениями. Это то же самое, что обратиться по первому и последнему индексу.

```
node.firstChild
node.lastChild
```

node.hasChildNodes()

Возвращает true или false, есть ли у элемента дочерние узлы или нет.

```
node.hasChildNodes()
проверить наличие дочерних узлов
```

node.contains

возвращает Boolean значение, указывающее, является ли узел потомком данного узла (все уровни вложенности):

```
node.contains( otherNode )
```

node.nextSibling

node.previousSibling

Обратиться к следующему и предыдущему соседу-узлу.

node.parentNode

Обратиться к родительскому узлу.

Навигация по тегам

для большинства задач нужно манипулировать узлами-элементами, которые формируют страницу.
API - Element

```
children
коллекция детей, которые являются элементами

firstElementChild, lastElementChild
первый и последний дочерний элемент

previousElementSibling, nextElementSibling
соседи-элементы

parentElement
родитель-элемент
```

.children

Коллекция детей-элементов

.firstElementChild

.lastElementChild

Первый и последний дочерний элемент

.previousElementSibling

.nextElementSibling

Обратиться к следующему и предыдущему соседу-элементу

.parentElement

Обратиться к родителю-элементу

Таблицы

Некоторые типы DOM-элементов предоставляют для удобства дополнительные свойства, специфичные для их типа. API по таблицам:
HTMLTableCaptionElement
HTMLTableCellElement
HTMLTableColElement
HTMLTableElement
HTMLTableRowElement
HTMLTableSectionElement

```
<table> дополнительно имеет
HTMLTableElement
Свойства
table.caption        возвращает заголовок таблицы.
table.tHead          возвращает header таблицы.
table.tFoot          возвращает footer таблицы.
table.rows           возвращает строки таблицы.
table.tBodies        возвращает тела таблицы.

table.align          возвращает/устанавливает выравнивание таблицы.
table.bgColor        возвращает/устанавливает цвет фона таблицы.
table.border         возвращает/устанавливает рамку таблицы.
table.cellPadding    возвращает/устанавливает cellpadding.
table.cellSpacing    возвращает/устанавливает cellspacing.
table.frame          определяет, какие стороны таблицы имеют рамку.
table.rules          определяет, какие внутренние рамки являются видимыми.
table.summary        возвращает/устанавливает описание таблицы.
table.width          возвращает/устанавливает ширину таблицы.

Методы()
table.createTHead    создает header таблицы.
table.deleteTHead    удаляет header таблицы.
table.createTFoot    создает footer таблицы.
table.deleteTFoot    удаляет footer таблицы.
```

```
table.insertRow      вставляет строку.
table.deleteRow      удаляет строку.
table.createCaption  создает заголовок таблицы.
table.deleteCaption  удаляет заголовок таблицы.


<tr>
HTMLTableRowElement
tr.cells             живая коллеция ячеек в ряду
tr.rowIndex          номер строки <tr> в таблице (включая все строки таблицы)
tr.sectionRowIndex   номер строки <tr> в текущей секции <thead>/<tbody>/<tfoot>.

Методы
tr.deleteCell()      удалить ячейку
tr.insertCell()      вставить ячейку

<thead>, <tfoot>, <tbody>

.rows

<td> <th>
.cellIndex     номер ячейки в строке <tr>
```

```
// выбрать первую таблицу в документе
a = document.getElementsByTagName('table')[0]

b = a.rows
// записать в переменную все tr, что есть в таблице

c = a.rows.cells[0]
// записать первую ячеку
```

Имена узлов и содержимое

Систематизация

```
Имя тега или узла
.nodeName     название тега или тип узла
.tagName      название тега, есть только у элементов Element
.nodeType     старомодный способ узнать тип узла

Класс узла
Можно узнать API таким образом
document.body.constructor.name  HTMLBodyElement
document.body.toString;         [object HTMLBodyElement]

Содержимое узлов
.nodeValue    текст внутри ноды. Можно менять. См. «Как вытащить текст из узлов»
.data         текст внутри ноды или элемента. Можно менять.

Содержимое элементов
.innerHTML    внутреннее HTML-содержимое узла-элемента. Можно менять.
.outerHTML    запись в elem.outerHTML не меняет elem. Можно менять.
.innerText    текстовое содержимое элемента и его потомков
.textContent  текст внутри узла-элемента (вычет всех тегов). Можно менять.
.hidden       true делает элемент невидимым, как style="display:none"

node.childElementCount  кол-во вложенных элементов узла

Встроенные свойства
.value   значение для <input>, <select> и <textarea> (HTMLInputElement, HTMLSelectElement…)
.href    адрес ссылки «href» для <a href="..."> (HTMLAnchorElement)
.id      значение атрибута «id» для всех элементов (HTMLElement)
```

.nodeType

Старомодный способ узнать «тип» DOM-узла. Его значением является цифра:
elem.nodeType == 1 для узлов-элементов,
elem.nodeType == 3 для текстовых узлов,
elem.nodeType == 9 для объектов документа.
В спецификации можно посмотреть остальные значения.

.nodeName

Получить имя ноды. Свойство есть у любых узлов.
Для элементов оно равно tagName в верхнем регистре, для остальных типов узлов (текст, комментарий и т.д.) оно содержит строку с типом узла.

.tagName

Свойство есть только у элементов Element. Возвращает имя в верхнем регистре.

node.childElementCount

Возвращает число дочерних **элементов** узла.
Можно применять к Document, DocumentFragment или Element

```
var elCount = Node.childElementCount;
```

Этого свойства нет в учебнике.

Изменение узлов

.innerHTML

Свойство innerHTML позволяет получить HTML-содержимое элемента в виде строки, полностью изменить содержимое или дополнить его. Свойство innerHTML есть только у узлов-элементов. У других типов узлов, в частности, у текстовых, есть свои аналоги: свойства nodeValue и data.

Получение содержимого:

```
<div>
    <p>Параграф внутри дива</p>
</div>

<script>

  let element = document.body.querySelector('div');

  console.log(element.innerHTML);
  // <p>Параграф внутри дива</p>

</script>
```

Перезапись содержимого
Свойство сначала удаляет всё содержимое элемента, потом перезаписывает с новым дополнением.
Так как содержимое «обнуляется» и переписывается заново, все изображения и другие ресурсы будут перезагружены.
Есть и другие побочные эффекты. Например, если существующий текст выделен мышкой, то при переписывании innerHTML большинство браузеров снимут выделение. А если это поле ввода <input> с текстом, введённым пользователем, то текст будет удалён. Есть и другие способы добавить содержимое, не использующие innerHTML (#? Какие?)

Скрипты не выполнятся.
Если innerHTML вставляет в документ тег <script> – он становится частью HTML, но не запускается.

Мы можем добавить новый HTML к элементу, используя
`elem.innerHTML+="ещё html"`

```
  <div>
    <p>Параграф внутри дива</p>
  </div>

  <script>
    let element = document.body.querySelector('div');

    element.innerHTML += "<p>Новый параграф</p>";

    console.log(element.innerHTML);
    // <p>Параграф внутри дива</p>
    // <p>Новый параграф</p>
  </script>
```

.innerText

Это свойство, позволяющее задавать или получать текстовое содержимое элемента и его потомков. MDN

Альтернативное свойство - .textContent, которое имеет ряд отличий:
- textContent получает содержимое *всех* элементов, включая <script> и <style>, тогда как innerText этого не делает.
- innerText умеет считывать стили и не возвращает содержимое скрытых элементов, тогда как textContent этого не делает.
- Метод innerText позволяет получить CSS, а textContent — нет.
- был введен Internet Explorer-ом

.outerHTML

Это как innerHTML, только к содержимому элемента добавляется ещё и сам элемент, на котором вызывается свойство:

```
<div>
  <p>Параграф внутри дива</p>
</div>

<script>
  let element = document.body.querySelector('div');
  console.log(element.outerHTML);

  // <div>
  // <p>Параграф внутри дива</p>
  // </div>

  element.outerHTML += '<p>Новый параграф</p>';
  // element хранит старое содержимое, которого больше нет в DOM, оно бесполезно.
  // вторую операцию нельзя провести с этим element, его просто нет в DOM.

  // При этом изменится HTML:
  // <div>...<div> <p>Новый параграф</p>


</script>
```

Дерево в браузере изменилось: в дереве появился такой же, но новый div и новый параграф.
При этом в переменной a всё ещё лежит старый div, который уже «оторван» от дерева.

произошло следующее:
- div был удалён из документа.
- Вместо него был вставлен другой HTML Новый элемент .
- В div осталось старое значение. Новый HTML не сохранён ни в какой переменной.

Поэтому легко сделать ошибку: заменить div.outerHTML, а потом продолжить работать с div, как будто там новое содержимое. Но это не так. Подобное верно для innerHTML, но не для outerHTML.
В отличие от innerHTML, запись в outerHTML не изменяет элемент. Вместо этого элемент заменяется целиком во внешнем контексте. Это потому, что использование outerHTML не изменяет DOM-элемент, а удаляет его из внешнего контекста и вставляет вместо него новый HTML-код.

node.nodeValue

Свойство Node.nodeValue возвращает или устанавливает значение текущего узла.
Применяется к текстовым узлам, комментариям и CDATA узлам.
Не применяется к узлам-элементам: всегда возвращает null.

```
value = node.nodeValue;
```

CharacterData.data

это DOMString, представляющая текстовые данные, которые содержит этот объект.
Применяется к текстовым узлам и комментариям.
Не применяется к элементам. См. «Как вытащить текст»

```
<body>
  Привет
  <!-- Комментарий -->

  <script>
    let text = document.body.firstChild;
    console.log(text.data);
    // Привет

    let comment = text.nextSibling;
    console.log(comment.data);
    // Комментарий

  </script>
</body>
```

Node.textContent

Позволяет задавать или получать текстовое содержимое элемента и его потомков.
Применяется именно к элементам и достаёт оттуда текст.
Не применяется к текстовым узлам в DOM.

```
<p>Параграф</p>

<script>
  let element = document.body.querySelector('p');
  console.log(element.textContent);
  // Параграф
</script>
```

Также позволяет записывать текст в textContent.
В отличие от innerHTML, вставка происходит именно «как текст», все символы трактуются буквально. Если написать , то будет вставлен не тег, а строка ‘ ’

Если внутри элемента содержатся другие элементы, то textContent вернёт весь текст из них:
Для узлов других типов (не элементов) textContent возвращает конкатенацию свойств textContent всех дочерних узлов, исключая комментарии и строки кода.

Если узел не имеет дочерних узлов, будет возвращена пустая строка. Установка данного значения удаляет все дочерние узлы и заменяет их единичным текстовым узлом с указанным значением.

```
<div id="news">
  <h1>Срочно в номер!</h1>
  <p>Марсиане атаковали человечество!</p>
</div>

<script>
  console.log(news.textContent);
  // Срочно в номер! Марсиане атаковали человечество!
</script>
```

Есть альтернативное свойство .innerText
Отличие от innerText
element.innerText был введен Internet Explorer-ом. Работает по тому же принципу за небольшими исключениями:
- textContent получает содержимое *всех* элементов, включая <script> и <style>. innerText этого не делает.
- innerText умеет считывать стили, не возвращает содержимое скрытых элементов. textContent этого не делает.
- innerText позволяет получить CSS. textContent — нет.
- innerText введён интернет эксплорером.

Отличие от innerHTML
Довольно часто innerHTML используется для получения или записи текста в элемент. Тем не менее, вместо него желательно использовать textContent: этот метод потребляет гораздо меньше ресурсов, так как текст парсится как текст, а не HTML. Кроме того, это защищает от XSS атак.

HTMLElement.hidden

Свойство является Boolean типом данных , который принимает значение true , если содержимое спрятано, в противном случае значение будет false.
Атрибут и DOM-свойство «hidden» указывает на то, видим ли мы элемент или нет.
Технически, `hidden` работает так же, как `style="display:none"`. Но его применение проще.

```
isHidden = HTMLElement.hidden;

HTMLElement.hidden = true | false;
```

Мы можем использовать его в HTML или назначать при помощи JavaScript:

```
<div>Оба тега DIV внизу невидимы</div>

<div hidden>С атрибутом "hidden"</div>

<div id="elem">С назначенным JavaScript свойством "hidden"</div>

<script>
  elem.hidden = true;
</script>
```

Мигающий элемент:

```
<div id="elem">Мигающий элемент</div>
  <script>
    setInterval(() => elem.hidden = !elem.hidden, 1000);
  </script>
```

Как вытащить текст из узлов и тегов

Для получения текста внутри узлов (в т.ч. элементов) и спользуются свойства .data и .nodeValue
На практике возникают проблемы, поэтому важно не забыть следующее:

```
<ul>
  <li id="li">Животные</li>
</ul>


Если применить .data к элементу, то ничего не получится, потому что текст – это потомок,
который хранится внутри элемента, а не его свойство.
li.data;  // undefined


Можно обратиться к текстовому узлу-потомку. В этом случае, браузер выведет именно объект, но не
не сам текст (даже если появятся символы). К этому результату нельзя применять методы строк или
конвертировать в строку.
li.childNodes[0];                   // { Животные } как объект
typeof li.childNodes[0];            // object
li.childNodes[0].toString();        // [object Text]
li.childNodes[0].constructor.name;  // Text


Чтобы получить именно внутренний текст, сначала получить дочерний текстовый узел, затем
применить методы.
li.childNodes[0].data;              // Животные
li.childNodes[0].nodeValue;         // Животные
```

HTML-атрибуты

Перечень

Есть две разные категории:
- атрибуты элементов, прописанные в HTML.
- свойства dom-объектов
О том, как они взаимосвязаны и как парсятся, написано в учебнике.

```
HTML-атрибуты
elem.attributes                    получить сразу все атрибуты элемента как коллекцию
elem.hasAttribute('name')          проверяет наличие атрибута.
elem.getAttribute('name')          получает значение атрибута.
elem.setAttribute('name', value)   устанавливает значение атрибута.
elem.removeAttribute('name')       удаляет атрибут

Запись нестандартных свойств в HTML
<body data-order-state="ok">

Получение нестандартных свойств в DOM
document.body.dataset.orderState
```

.attributes

Свойство Element.attributes возвращает группу атрибутов всех узлов, зарегистрированных в указанном узле.
Это коллекция объектов NamedNodeMap, которая принадлежит ко встроенному классу Attr со свойствами name и value. Можно обратиться к его элементам по индексу: elem.attributes[0]. MDN

```
let attr = element.attributes;
```

Коллекция attributes является перебираемой. Все атрибуты элемента становятся объектами со свойствами name и value.

.getAttribute()
Возвращает значение указанного атрибута элемента. Если элемент не содержит данный атрибут, могут быть возвращены null или "" (пустая строка). Имена атрибутов регистронезависимы. MDN

```
var attribute = element.getAttribute(attributeName);
```

.setAttribute(name, value)
Добавляет новый атрибут или изменяет значение существующего атрибута у выбранного элемента. MDN

```
element.setAttribute(name, value);
```

name - имя атрибута (строка).
value - значение атрибута.

.removeAttribute()
удаляет аттрибут с элемента. MDN

```
element.removeAttribute(attrName);


// <div id="div1" align="left" width="200px">
document.getElementById("div1").removeAttribute("align");
// now: <div id="div1" width="200px">
```

.hasAttribute()
Вовзращает Boolean, указывающее, имеет элемент определенный атрибут, или нет. MDN

```
var result = element.hasAttribute(attName);
```

result – хранит возвращенное значение true или false.
attName – это String представляющая имя атрибута.

.dataset
Когда браузер парсит HTML, он распознаёт стандартные атрибуты и создаёт DOM-свойства для них. Если атрибут нестандартный, то он не преобразуется в свойство объекта DOM.

Чтобы нестандартные атрибуты парсились, надо записывать их в html с префикса «data-».
Такие атрибуты автоматом станут доступны в свойстве DOM-объекта .dataset.

```
Запись нестандартных свойств в HTML
<body data-order-state="ok">

Получение нестандартных свойств в DOM
document.body.dataset.orderState
```

Атрибуты, состоящие из нескольких слов, data-order-state, становятся свойствами, записанными с помощью верблюжьей нотации: .dataset.orderState.
MDN не нашёл.

Изменение DOM

Перечень

```
Создать узел
document.createElement('element') let div = document.createElement('div');
document.createTextNode('text')   let textNode = document.createTextNode('А вот и я');

Копирование узлов
elem.cloneNode(true)   клон элемента со всеми атрибутами и дочерними элементами
elem.cloneNode(false)  клон без дочерних элементов

Замена узлов
node.replaceWith(...nodes or strings)    заменяет node заданными узлами или строками

Удаление узлов
node.remove()


Безопасная вставка узлов и строк
Можно создать элемент, сохранить в переменной и вставить как ноду, но не как html-код
node.append(...nodes or strings)    добавляет узлы или строки в конец node
node.prepend(...nodes or strings)   вставляет узлы или строки в начало node
node.before(...nodes or strings)    вставляет узлы или строки до node
node.after(...nodes or strings)     вставляет узлы или строки после node


Вставка HTML кода (элементов) и его братья
elem.insertAdjacentHTML(where, html)     универсальный метод по вставке кода html
elem.insertAdjacentText(where, text)     вставить текст
elem.insertAdjacentElement(where, elem)  вставить готовую ноду

where
"beforebegin"   вставить html непосредственно перед elem
"afterbegin"    вставить html в начало elem
"beforeend"     вставить html в конец elem
"afterend"      вставить html непосредственно после elem

html
строка, которая будет вставлена как код HTML


Устаревшие методы
parentElem.appendChild(node)
добавляет node в конец дочерних элементов parentElem

parentElem.insertBefore(node, nextSibling)
вставляет node перед nextSibling в parentElem

parentElem.replaceChild(node, oldChild)
заменяет oldChild на node среди дочерних элементов parentElem

parentElem.removeChild(node)
удаляет node из parentElem (предполагается, что он родитель node).

let fragment = new DocumentFragment()
document.write('<b>Привет</b>');
```

Создание элементов

document.createElement()
Создаёт новый элемент с заданным тегом. MDN

```
var element = document.createElement(tagName, [options]);

let div = document.createElement('div');
```

element — созданый объект элемента.
tagName — строка, указывающая элемент какого типа должен быть создан.
options — необязательный параметр, объект ElementCreationOptions, который может содержать только поле is, указывающее имя пользовательского элемента, созданного с помощью customElements.define() (см. Веб-компоненты).

document.createTextNode()
Создаёт новый *текстовый узел* с заданным текстом. MDN

```
var text = document.createTextNode(data);

let textNode = document.createTextNode('А вот и я');
```

text - это текстовый узел.
data - это строка с данными, которые будут помещены в текстовый узел.

Вставка узлов
Эти методы могут использоваться только для вставки DOM-узлов или текстовых фрагментов, не для элементов. Они работают как .textContent. Можно создать элемент, сохранить в переменной и вставить его, только тогда он будет вставлен как элемент.
Эти методы могут вставлять несколько узлов и текстовых фрагментов за один вызов.

Методы относятся к нескольким API:
ParentNode MDN
ChildNode MDN

ParentNode.append
Добавляет узлы или строки в конец node (после последнего потомка). Почему-то не могу найти документацию.

```
node.append(...nodes or strings)
```

Можно одновременно добавлять сразу несколько узлов или строк:

```
const div = document.createElement('div');
div.innerHTML = "<strong>text inside div</strong>";

const p = document.createElement('p');
p.innerHTML = "text inside p";

document.body.append(div, p);
```

Отличия от Node.appendChild():
- позволяет добавлять DOMString, а Node.appendChild() принимает только Node.
- ничего не возвращает, а Node.appendChild() возвращает добавленный объект Node.
- можно добавить несколько узлов (node) или строк, а Node.appendChild() может добавть только один узел.

ParentNode.prepend
Метод вставляет множество объектов Node или DOMString в начало (перед первым потомком) ParentNode.

```
ParentNode.prepend(...nodes or strings)
```

ChildNode.before
вставляет узлы или строки до node

```
ChildNode.before(...nodes or strings)
```

ChildNode.after
вставляет узлы или строки после node

```
ChildNode.after(...nodes or strings)
```

ChildNode.replaceWith
заменяет node заданными узлами или строками

```
ChildNode.replaceWith(...nodes or strings)
```

Методы вставки элементов
Используется для вставки HTML со всеми тегами, как это делает .innerHTML.
API - Element
На практике часто используется только insertAdjacentHTML. Потому что для элементов и текста у нас есть методы append/prepend/before/after – их быстрее написать, и они могут вставлять как узлы, так и текст.

position
'beforebegin' до самого element (до открывающего тега).
'afterend' после element (после закрывающего тега).
'afterbegin' сразу после открывающего тега element (перед первым потомком).
'beforeend' сразу перед закрывающим тегом element (после последнего потомка).

elem.insertAdjacentHTML()
Вставить строку именно как html.
Разбирает указанный текст как HTML или XML и вставляет полученные узлы (nodes) в DOM дерево в указанную позицию. Данная функция не переписывает имеющиеся элементы, что предотвращает дополнительную сериализацию и поэтому работает быстрее, чем манипуляции с innerHTML.

```
targetElement.insertAdjacentHTML(position, text);
```

elem.insertAdjacentText()
Метод помещает заданный текстовый узел в указанную позицию.
Текст не будет интерпретироваться как HTML.

```
element.insertAdjacentText(position, element);
```

elem.insertAdjacentElement()
Добавляет уже готовый элемент в DOM-дерево.
Метод возвращает добавляемый элемент, либо null, если добавление элемента завершилось ошибкой.

```
targetElement.insertAdjacentElement(position, element);
```

Удаление и замена узлов

ChildNode.remove()
Удаление ChildNode узла из DOM дерева.

Сам объект остаётся и исчезнет только тогда, когда исчезнут все ссылки на него.

ChildNode.after() - ещё раз

берёт ChildNode и после него вставляет переданный узел.
Параграфы поменяются местами:

```
<p id='first'>first</p>
<p id='second'>seond</p>

<script>
  second.after(first);
</script>
```

Node.cloneNode()

Возвращает дубликат узла, из которого этот метод был вызван. MDN
Если понадобится сделать второй подобный существующему элемент, его можно не создавать с нуля, а клонировать и немного изменить текст внутри.

```
var dupNode = node.cloneNode(deep);
```

node
Узел, который будет клонирован.

dupNode
Новый узел, который будет клоном node

deep Необязательный
true, если дети узла должны быть клонированы
false для того, чтобы был клонирован только указанный узел.

```
<div class="alert" id="div">
  <strong>Всем привет!</strong> Вы прочитали важное сообщение.
</div>

<script>
  let div2 = div.cloneNode(true);
  // клонировать сообщение
  div2.querySelector('strong').innerHTML = 'text2';
  // изменить клонированный элемент

  div.after(div2);
  // показать клонированный элемент после существующего div
</script>
```

DocumentFragment - обёртка

Это узел-обёртка, в который можно поместить другие узлы, как в хранилище, а потом куда-то вставить. При вставке обёртка исчезает и остаются только те элементы, которые были внутри.
Узлы в него закидываются через метод .append
Много методов на странице документации MDN.

Интерфейс DocumentFragment представляет собой минимальный объект документа, который не имеет родителя. Он используется как легкая версия Document, чтобы хранить фрагменты XML.
Различные другие методы могут взять document fragment в качестве аргумента (например, любые методы интерфейса Node, такие как Node.appendChild и Node.insertBefore), в этом случае прикрепляются или вставляются дети фрагмента, а не сам фрагмент.

Эта функция используется редко, потому что можно все элементы закинуть в обычный массив и распаковать их через rest. DocumentFragment используется в некоторых областях: например, для элемента template.

Пустой DocumentFragment создаётся с помощью метода document или конструктора:

```
Document.createDocumentFragment()
new DocumentFragment()
```

Узлы в него закидываются через метод **.append**

```
<ul id='ul'></ul>

<script>
  let elementLi = document.createElement('li');
  let foo = new DocumentFragment();
  foo.append(elementLi); // теперь li внутри контейнера
  ul.append(foo);
</script>
```

Функция в примере генерирует фрагмент с элементами <li>, которые позже вставляются в <ul>. Это замена DocumentFragment:

```
<ul id="ul"></ul>

<script>
function getListContent() {
  let result = [];

  for(let i=1; i<=3; i++) {
    let li = document.createElement('li');
    li.append(i);
    result.push(li);
  }

  return result;
}

ul.append(...getListContent());
</script>
```

Устаревшие методы вставки/удаления

Маловероятно, что они устаревшие. Учебник их не рекомендует к использованию в новых скриптах, но в руководстве об этом ни слова. Наоборот, «новые» методы руководство считает экспериментальными и не рекомендует к использованию.

Все эти методы возвращают вставленный/удалённый узел. Но обычно возвращаемое значение не используют, просто вызывают метод.

Node.appendChild()

Добавляет node в конец дочерних элементов parentElem.
Возвращает добавленный узел.

```
var child = element.appendChild(child);
```

element родительский элемент.

child это элемент вставляется в конец element.

Добавляет узел в конец списка дочерних элементов указанного родительского узла. Если данный дочерний элемент является ссылкой на существующий узел в документе, то функция appendChild() перемещает его из текущей позиции в новую позицию (нет необходимости удалять узел из родительского узла перед добавлением его к какому-либо другому узлу).

Это означает, что узел не может находиться в двух точках документа одновременно. Поэтому, если у узла уже есть родитель, он сначала удаляется, а затем добавляется в новую позицию. Node.cloneNode() можно использовать для создания копии узла перед добавлением его в новый родительский элемент.

Node.insertBefore()

добавляет элемент в список дочерних элементов родителя перед указанным элементом.
Возвращает вставляемый элемент.

```
var insertedElement = parentElement.insertBefore(newElement, referenceElement);
```

insertedElement	Вставленный элемент.
parentElement	Родитель для нового элемента.
newElement	Элемент для вставки.
referenceElement	Элемент, перед которым будет вставлен newElement.

Node.replaceChild()

Заменяет дочерний элемент на выбранный.
Возвращает замененный элемент.

```

```

Node.removeChild(node)
Удаляет node из parentElem (предполагается, что он родитель node).

```
replacedNode = parentNode.replaceChild(newChild, oldChild);
```

newChild
элемент на который будет заменен oldChild. В случает если он уже есть в DOM, то сначала он будет удален.
oldChild
элемент который будет заменен.
replacedNode
замененный элемент. Тоже самое что и oldChild.

Document.write()

ещё один древний метод добавления содержимого на веб-страницу.
Вызов document.write работает только во время загрузки страницы. Если вызвать его позже, то существующее содержимое документа затрётся. Так что после того, как страница загружена, он уже непригоден к использованию, в отличие от других методов DOM.

MDN
Пишет строку в поток документа, открытый с помощью document.open().
Замечание: поскольку document.write пишет строку в **поток** документа, вызов document.write для закрытого (но загруженного) документа автоматически вызовет document.open, который очистит документ.

Новый конспект

Браузерное окружение, спецификации

Окружение предоставляет свои объекты и дополнительные функции в дополнение к базовым языковым. Браузеры, например, дают средства для управления веб-страницами. Node.js делает доступными какие-то серверные возможности и так далее.

В браузерном окружении корневым объектом является window, который предоставляет как базовые возможности js, так и дополнительные для работы с окном браузера и документом HTML:

```
           window
             |
   DOM      BOM     JavaScript
    |        |          |
document  screen     Object
          history    Array
```

DOM – это Document Object Model, объектная модель документа. Это программный интерфейс (API), позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML- и XML-документов и изменять его.
Он представляет всё содержимое страницы в виде объектов, которые можно менять. Объект document – основная «входная точка», с которого начинается весь документ. Это означает, что в структуре DOM-дерева, document является вершиной дерева.
Описывает структуру документа, манипуляции с контентом и события.
Документация по DOM находится на whatwg и в MDN.

BOM – это Browser Object Model, объектная модель браузера. Она предоставляет инструменты дря работы с браурезом (со всем, кроме документа). Это набор глобальных объектов, управляющих поведением именно браузера. Они предоставляются чтобы работать со всем, кроме документа.
Эти глобальные объкты также находятся внутри window. Например, функции alert/confirm/prompt тоже являются частью BOM. Например, посмотреть историю посещений, операционную систему, размеры окна и т.д. Является частью спецификации HTML.
Документация по BOM находится на whatwg и в MDN.

Спецификация HTML – Описывает язык HTML и BOM. setTimeout, alert, location – это всё отсюда.

CSSOM – описывает файлы стилей, правила написания стилей и манипуляций с ними, а также то, как это всё связано со страницей. Используется редко, упоминаю тут для общего развития.
Документация на w3.

DOM-дерево

В соответствии с объектной моделью документа, каждый HTML-тег является объектом. Вложенные теги являются «детьми» родительского элемента. Текст, который находится внутри тега, также является объектом.
DOM – это представление HTML-документа в виде дерева тегов. Все эти объекты доступны при помощи JavaScript.

Html и DOM – не одно и то же. HTML – это разметка. На основании этой разметки браузер построит соответствующие объекты, с которыми сможет работать JS. Это обеспечивает структурированное представление документа (дерева). DOM обеспечивает представление документа в виде структурированной группы узлов и объектов, которые имеют свойства и методы. По сути, она связывает веб-страницы со скриптами или языками программирования.
Все библиотеки (jquery и другие) и фреймворки (angular, react) внутри себя манипулируют DOM. Это та база, вокруг которой построено всё во фронтенд разработке. DOM-tree с точки зрения javascript представляет из себя объект.

В общих чертах процесс отображения страницы можно представить следующим образом:

1. Браузер выясняет адрес сервера с помощью DNS
2. Браузер выполняет запрос на сервер
3. Пришедший в ответ html парсится и на его основе строится DOM дерево
4. Браузер рисует страницу, используя DOM дерево.

Итак, DOM-дерево состоит из узлов (нод). Существует 12 типов узлов. Но на практике мы в основном работаем с 4 из них:
1. элементы HTML
2. текст
3. комментарии
4. document – «входная точка» в DOM

Навигация по DOM

```
node
.childNodes
.firstChild
.lastChild
.hasChildNodes() функция проверки дочерних узлов

.nextSibling
.previousSibling.

.parentNode

element
.children
.firstElementChild
.lastElementChild

.nextElementSibling
.previousElementSibling.

.parentElement

Верхушка
<html> = document.documentElement
<head> = document.head
<body> = document.body
```

Все операции с DOM начинаются с объекта document. Это главная «точка входа» в DOM. Из него мы можем получить доступ к любому узлу.

```
                  document
                     |
                   <html>
          document.documentElement
              |            |
           <body>       <head>
      document.body   document.head



                 parrentNode
                     |
previousSibling - <div> - nextSibling
                     |
                 childNodes
                     |
         firstChild     lastChild
```

Дочерние узлы (дети) – элементы, которые являются непосредственными детьми узла. Другими словами, элементы, которые лежат непосредственно внутри данного. Например, <head> и <body> являются детьми элемента <html>.
Потомки – все элементы, которые лежат внутри данного, включая детей, их детей и т.д.

Ноды

Узлы – это вообще всё, что есть в дереве: элементы, текст и комменты.

```
.childNodes
.firstChild
.lastChild
.hasChildNodes() функция проверки дочерних узлов

.nextSibling
.previousSibling.

.parentNode
```

Коллекция childNodes содержит список всех детей, включая текстовые узлы. Для перебора можно использовать цикл for..of. Можно конвертировать в массив: Array.from(document.body.childNodes)

DOM-коллекции и *все* навигационные свойства доступны только для чтения.
Мы не можем заменить один дочерний узел на другой, просто написав childNodes[i] = ... Для изменения DOM требуются другие методы.

DOM-коллекции живые
Почти все DOM-коллекции, за небольшим исключением, живые. Другими словами, они отражают текущее состояние DOM. Если мы сохраним ссылку на elem.childNodes и добавим/удалим узлы в DOM, то они появятся в сохранённой коллекции автоматически.

Элементы

Это только узлы, которые являются html-элементами.

```
.children
.firstElementChild
.lastElementChild

.nextElementSibling
.previousElementSibling.

.parentElement
```

Таблицы

Некоторые типы DOM-элементов предоставляют для удобства дополнительные свойства, специфичные для их типа. API по таблицам:
HTMLTableCaptionElement
HTMLTableCellElement
HTMLTableColElement
HTMLTableElement
HTMLTableRowElement
HTMLTableSectionElement

```
<table> дополнительно имеет
HTMLTableElement
Свойства
table.caption  возвращает заголовок таблицы.
table.tHead    возвращает header таблицы.
table.tFoot    возвращает footer таблицы.
```

```
table.rows     возвращает строки таблицы.
table.tBodies  возвращает тела таблицы.

table.align        возвращает/устанавливает выравнивание таблицы.
table.bgColor      возвращает/устанавливает цвет фона таблицы.
table.border       возвращает/устанавливает рамку таблицы.
table.cellPadding  возвращает/устанавливает cellpadding.
table.cellSpacing  возвращает/устанавливает cellspacing.
table.frame        определяет, какие стороны таблицы имеют рамку.
table.rules        определяет, какие внутренние рамки являются видимыми.
table.summary      возвращает/устанавливает описание таблицы.
table.width        возвращает/устанавливает ширину таблицы.

Методы()
table.createTHead    создает header таблицы.
table.deleteTHead    удаляет header таблицы.
table.createTFoot    создает footer таблицы.
table.deleteTFoot    удаляет footer таблицы.
table.insertRow      вставляет строку.
table.deleteRow      удаляет строку.
table.createCaption  создает заголовок таблицы.
table.deleteCaption  удаляет заголовок таблицы.


<tr>
HTMLTableRowElement
tr.cells             живая коллеция ячеек в ряду
tr.rowIndex          номер строки <tr> в таблице (включая все строки таблицы)
tr.sectionRowIndex   номер строки <tr> в текущей секции <thead>/<tbody>/<tfoot>.

Методы
tr.deleteCell()      удалить ячейку
tr.insertCell()      вставить ячейку

<thead>, <tfoot>, <tbody>

.rows

<td> <th>
.cellIndex     номер ячейки в строке <tr>

// выбрать первую таблицу в документе
a = document.getElementsByTagName('table')[0]

b = a.rows
// записать в переменную все tr, что есть в таблице

c = a.rows.cells[0]
// записать первую ячеку
```

Поиск по DOM

Это получение произвольных элементов без перемещения по дереву.

```
document.getElementById('id')

elem.querySelectorAll(css)
elem.querySelector(css)

elem.getElementsByClassName(className)
document.getElementsByName(name)
elem.getElementsByTagName(tag)    * вместо тега вернёт всех потомков

elem.closest(css)     ближайший соответствующий предок
elem.matches(css)     проверяет, удовлетворяет ли elem CSS-селектору
elemA.contains(elemB)  проверяет на наличие потомка внутри
```

Все методы "getElementsBy*" возвращают живую коллекцию.
querySelectorAll возвращает статическую коллекцию.

В то же время, путь любого элемента для js можно найти в консоли через выбор элемента - правая кнопка мыши - copy - copy js path

Свойства узлов: тип, тег и содержимое

```
Имя тега или узла
.nodeName  название тега или тип узла
.tagName   название тега, есть только у элементов Element
.nodeType  старомодный способ узнать тип узла

Класс узла
Можно узнать API таким образом
document.body.constructor.name  HTMLBodyElement
document.body.toString;         [object HTMLBodyElement]

Содержимое узлов
.nodeValue    текст внутри ноды. Можно менять.
.data         текст внутри ноды или элемента. Можно менять.

Содержимое элементов
.innerHTML    внутреннее HTML-содержимое узла-элемента. Можно менять.
.outerHTML    запись в elem.outerHTML не меняет elem. Можно менять.
.innerText    текстовое содержимое элемента и его потомков
.textContent  текст внутри узла-элемента (вычет всех тегов). Можно менять.
.hidden       true делает элемент невидимым, как style="display:none"

node.childElementCount  кол-во вложенных элементов узла

Собственные свойства
.value   значение для <input>, <select> и <textarea> (HTMLInputElement, HTMLSelectElement…)
.href    адрес ссылки «href» для <a href="..."> (HTMLAnchorElement)
.id      значение атрибута «id» для всех элементов (HTMLElement)
```

У узлов есть общие и специальные свойства в зависимости от их типа. Например, у <a> - специальные свойства, связанные с сылками.
Каждый DOM-узел принадлежит соответствующему встроенному классу. Корнем иерархии является EventTarget (обеспечивает поддержку событий), от него наследует Node (parentNode, nextSibling, childNodes), от него – Element (getElementsByTagName, querySelector, nextElementSibling), от него HTMLElement и остальные DOM-узлы. Полный набор свойств и методов конкретного узла собирается в результате наследования.

```
Object      hasOwnProperty

EventTarget обеспечивает поддержку событий
  |
Node        parentNode, nextSibling, childNodes
  |
Element     getElementsByTagName, querySelector, nextElementSibling
  |
HTMLElement
```

console.log(elem) выводит элемент в виде DOM-дерева.
console.dir(elem) выводит элемент в виде DOM-объекта, что удобно для анализа его свойств.

Узнать тип узла

Узнать имя **класса** DOM-узла

```
document.body.constructor.name    // HTMLBodyElement
document.body.toString;           // [object HTMLBodyElement]


nodeType старомодный способ узнать тип узла
elem.nodeType == 1 узел-элемент
elem.nodeType == 3 текст
elem.nodeType == 9 объекты документа
```

Проверить наследование можно также при помощи instanceof.

Получив DOM-узел, мы можем узнать **имя его тега** из свойств nodeName и tagName.
Имена тегов (кроме XHTML) всегда пишутся в верхнем регистре

```
alert( document.body.nodeName ); // BODY
alert( document.body.tagName );  // BODY
```

tagName есть только у элементов Element
nodeName определено для любых узлов Node

Содержимое элементов

innerHTML
содержимое внутри элемента в виде строки. Есть только у узлов-элементов. Его можно изменять, и даже если написать +=, то это всегда будет именно перезапись.
Скрипты не выполнятся.
Если innerHTML вставляет в документ тег <script> – он становится частью HTML, но не запускается.

```
document.body.innerHTML;
document.body.innerHTML = 'Новый BODY!';
```

Так как содержимое «обнуляется» и переписывается заново, все изображения и другие ресурсы будут перезагружены.
Есть и другие побочные эффекты. Например, если существующий текст выделен мышкой, то при переписывании innerHTML большинство браузеров снимут выделение. А если это поле ввода <input> с текстом, введённым пользователем, то текст будет удалён.
Есть и другие способы добавить содержимое.

outerHTML
Перезапись HTML элемента целиком. Это как innerHTML плюс сам элемент. В отличие от innerHTML, запись в outerHTML не изменяет элемент. Вместо этого элемент заменяется целиком во внешнем контексте.

.nodeValue
node.data
содержимое текстового узла. См. «Как вытащить текст»
Текст, который содержится в элементе. Например, текст в text

```
text.data
comment.data
li.data
```

textContent
просто текст. Предоставляет доступ к тексту внутри элемента за вычетом всех <тегов>.
Возвращает только текст, как если бы все <теги> были вырезаны, но текст в них остался.
Записывает текст (безопасно) внутрь тега.

```

```

.hidden
Делает элемент невидимым.
Технически, hidden работает так же, как style="display:none". Но его применение проще.

```
elem.hidden = true;
```

Другие свойства
Большинство стандартных HTML-атрибутов имеют соответствующее DOM-свойство, и мы можем получить к нему доступ.
Если же нам нужно быстро что-либо узнать или нас интересует специфика определённого браузера – мы всегда можем вывести элемент в консоль, используя console.dir(elem), и прочитать все свойства.

```
.value   значение для <input>, <select> и <textarea> (HTMLInputElement, HTMLSelectElement…)
.href    адрес ссылки «href» для <a href="..."> (HTMLAnchorElement)
.id      значение атрибута «id» для всех элементов (HTMLElement)
```

! Как вытащить текст из узлов и тегов

Для получения текста внутри узлов (в т.ч. элементов) и спользуются свойства .data и .nodeValue
На практике возникают проблемы, поэтому важно не забыть следующее:

```
<ul>
  <li id="li">Животные</li>
</ul>


Если применить .data к элементу, то ничего не получится, потому что текст – это потомок,
который хранится внутри элемента, а не его свойство.
li.data;  // undefined


Можно обратиться к текстовому узлу-потомку. В этом случае, браузер выведет именно объект, но не
не сам текст (даже если появятся символы). К этому результату нельзя применять методы строк или
конвертировать в строку.
li.childNodes[0];                   // { Животные } как объект
typeof li.childNodes[0];            // object
li.childNodes[0].toString();        // [object Text]
li.childNodes[0].constructor.name;  // Text


Чтобы получить именно внутренний текст, сначала получить дочерний текстовый узел, затем
применить методы.
li.childNodes[0].data;              // Животные
li.childNodes[0].nodeValue;         // Животные
li.childNodes[0].textContent;       // Животные
```

Считаем потомков

```
let list = document.querySelector('ul');

function count(ul) {
```

```
  let children = ul.children;

  for(let li of children) {
    let liName = li.firstChild.data.trim();
    let numberOfInnerUl = 0;
    let arrayOfUl = [];

    for (let node of li.children) {
      if (node.tagName === 'UL') {
        numberOfInnerUl += node.children.length;
        arrayOfUl.push(node);
      }
    }

    if (numberOfInnerUl) {
      console.log(liName, numberOfInnerUl);
      arrayOfUl.forEach((ul) => count(ul));
    } else {
      console.log(liName);
    }

  }
}

count(list);
```

Атрибуты и свойства

Для узлов-элементов большинство стандартных HTML-атрибутов автоматически становятся свойствами DOM-объектов. Например, body.id="page". Такое преобразование атрибута элемента в свойство объекта происходит в большинстве случаев.

атрибуты vs свойства

Свойства объектов

DOM-узлы – это обычные объекты JavaScript. В объекты (именно в объекты, а не в HTML-теги) можно дописывать собственные свойства:

```
// запись нового свойства
document.body.myProperty = 123;

console.log(document.body.myProperty);
// 123
```

Также можно изменять встроенные прототипы и добавлять новые методы ко всем элементам:

```
Element.prototype.sayHi = function() {
  alert(`Hello, I'm ${this.tagName}`);
};

document.documentElement.sayHi();  // Hello, I'm HTML
document.body.sayHi();             // Hello, I'm BODY
```

Атрибуты

Когда браузер парсит HTML, чтобы создать DOM-объекты для тегов, он распознаёт стандартные атрибуты и создаёт DOM-свойства для них. Браузер НЕ распознаёт нестандартные атрибуты.
Если атрибут не стандартный и записан в html-тег, то нестандартный атрибут не преобразуется в свойство DOM-объекта:

```
<body something="non-standard">

<script>
    console.log(document.body.something); // undefined
</script>
```

Важно, что нестандартный атрибут остаётся в HTML и его всё ещё можно получить методами, которые работают с html.

Чтобы нестандартные атрибуты распарсились в свойства dom-объектов, надо использовать префикс «data-». О нём написано ниже. Такие атрибуты автоматом станут доступны в свойстве DOM-объекта .dataset.

```
<body data-about="Elephants">
<script>
  alert(document.body.dataset.about); // Elephants
</script>
```

Ещё несколько особенностей атрибутов:
- их значения – это всегда строки. Можно присвоить что угодно, это станет строкой.
- имена атрибутов регистронезависимы.
- все атрибуты видны в .outerHTML.

Доступ ко всем атрибутам

Получить доступ и к стандартным, и к пользовательским атрибутам можно с помощью этих методов:

```
elem.hasAttribute('name')          проверяет наличие атрибута.
elem.getAttribute('name')          получает значение атрибута.
elem.setAttribute('name', value)   устанавливает значение атрибута.
elem.removeAttribute('name')       удаляет атрибут

elem.attributes                    получить сразу все атрибуты элемента
```

- Коллекция attributes является перебираемой. Все атрибуты элемента становятся объектами со свойствами name и value.
- Если же нужно значение href или любого другого атрибута в точности, как оно записано в HTML, можно воспользоваться getAttribute.

Эти методы работают именно с тем, что написано в HTML, поэтому они могут читать нестандартные атрибуты:

```
<body something="123">

<script>
    console.log(document.body.getAttribute('something')); // 123
</script>
```

Если атрибут состоит из нескольких слов в html, то всё так и записывается (в отличие от dom, где запись происходит через верблюда):

```
elem.getAttribute('data-widget-name')
```

Синхронизация

Когда стандартный атрибут изменяется, соответствующее свойство автоматом обновляется, и наоборот.
Но есть и исключения, например, input.value синхронизируется только в одну сторону: атрибут => значение.

Особенности DOM-свойств

DOM-свойства типизированны и не всегда являются строками.
Например, свойство input.checked (для чекбоксов) имеет логический тип:

```
<input id="input" type="checkbox" checked>

<script>
  console.log(input.getAttribute('checked'));  // пустая строка
  console.log(input.checked);                  // true
</script>
```

Атрибут style – строка, но свойство style является объектом:

```
<div id="div" style="color:red;font-size:120%">Hello</div>

<script>

console.log(div.getAttribute('style'));
// строка color:red;font-size:120%
console.log(div.style);
// [object CSSStyleDeclaration]
console.log(div.style.color);
// red

</script>
```

DOM-свойство href всегда содержит полный URL, даже если атрибут содержит относительный URL или просто #hash. Если же нужно значение href или любого другого атрибута в точности, как оно записано в HTML, можно воспользоваться getAttribute.

data-

Чтобы избежать конфликтов нестандартных и стандартных атрибутов, все нестандартные атрибуты надо записывать в html с префикса «data-».
Такие атрибуты автоматом распарсятся и станут доступны в свойстве DOM-объекта .dataset.

```
data-*
Определяет группу атрибутов, называемых атрибутами пользовательских данных, позволяющих
осуществлять обмен служебной информацией между HTML и его DOM представлением.

HTMLOrForeignElement.dataset
Предоставляет доступ как для чтения, так и для изменения пользовательских data-атрибутов (data-
*). Это map of DOMString, одна запись для каждого пользовательского атрибута данных.

Само свойство dataset доступно только для чтения. Для записи должны использоваться  его
"свойства", которые представлены data-атрибутами.


<body data-about="Elephants">
<script>
  alert(document.body.dataset.about); // Elephants
</script>
```

Атрибуты, состоящие из нескольких слов, data-order-state, становятся свойствами, записанными с помощью верблюжьей нотации: .dataset.orderState.

Пример использования нестандартных атрибутов

Иногда нестандартные атрибуты используются для передачи пользовательских данных из HTML в JavaScript, или чтобы «помечать» HTML-элементы для JavaScript.

```
<!-- пометить div, чтобы показать здесь поле "name" -->
<div show-info="name"></div>
<!-- а здесь возраст "age" -->
<div show-info="age"></div>
```

```
<script>
  // код находит элемент с пометкой и показывает запрошенную информацию
  let user = {
    name: "Pete",
    age: 25
  };

  for(let div of document.querySelectorAll('[show-info]')) {
    // вставить соответствующую информацию в поле
    let field = div.getAttribute('show-info');
    div.innerHTML = user[field];
    // сначала Pete в name, потом 25 в age
  }
</script>
```

Изменение документа

Перечень

```
Создать узел
document.createElement('element') let div = document.createElement('div');
document.createTextNode('text')   let textNode = document.createTextNode('А вот и я');

Копирование узлов
elem.cloneNode(true)   клон элемента со всеми атрибутами и дочерними элементами
elem.cloneNode(false)  клон без дочерних элементов

Замена узлов
node.replaceWith(...nodes or strings)    заменяет node заданными узлами или строками

Удаление узлов
node.remove()


Безопасная вставка узлов и строк
Можно создать элемент, сохранить в переменной и вставить как ноду, но не как код
node.append(...nodes or strings)    добавляет узлы или строки в конец node
node.prepend(...nodes or strings)   вставляет узлы или строки в начало node
node.before(...nodes or strings)    вставляет узлы или строки до node
node.after(...nodes or strings)     вставляет узлы или строки после node


Вставка HTML кода (элементов) и его братья
elem.insertAdjacentHTML(where, html)     универсальный метод по вставке кода html
elem.insertAdjacentText(where, text)     вставить текст
elem.insertAdjacentElement(where, elem)  вставить готовую ноду

where
"beforebegin"   вставить html непосредственно перед elem
"afterbegin"    вставить html в начало elem
"beforeend"     вставить html в конец elem
"afterend"      вставить html непосредственно после elem

html
строка, которая будет вставлена как код HTML


Устаревшие методы
parentElem.appendChild(node)
добавляет node в конец дочерних элементов parentElem

parentElem.insertBefore(node, nextSibling)
вставляет node перед nextSibling в parentElem
```

```
parentElem.replaceChild(node, oldChild)
заменяет oldChild на node среди дочерних элементов parentElem

parentElem.removeChild(node)
удаляет node из parentElem (предполагается, что он родитель node).

let fragment = new DocumentFragment()
document.write('<b>Привет</b>');
```

Создание узлов

```
document.createElement('element')
let div = document.createElement('div');

document.createTextNode('text')
let textNode = document.createTextNode('А вот и я');
```

Вставка, перемещение узлов и строк

Эти методы могут вставлять несколько узлов или строк за раз.
Можно вставлять вновь созданные элементы или перемещать существующие.
Строки вставляются безопасным способом именно как текст.

```
node.append(...nodes or strings)         добавляет узлы или строки в конец node
node.prepend(...nodes or strings)        вставляет узлы или строки в начало node
node.before(...nodes or strings)         вставляет узлы или строки до node
node.after(...nodes or strings)          вставляет узлы или строки после node
node.replaceWith(...nodes or strings)    заменяет node заданными узлами или строками
```

Пример использования

```
let liFirst = document.createElement('li');
liFirst.innerHTML = 'prepend';
ol.prepend(liFirst); // вставить liFirst в начало <ol>

document.body.append(div)

div.before('<p>Привет</p>', document.createElement('hr'));
```

Вставка HTML и текста

```
elem.insertAdjacentHTML(where, html)
это универсальный метод по вставке html

elem.insertAdjacentText(where, text)
elem.insertAdjacentElement(where, elem)
это ещё два до кучи

where
"beforebegin"   вставить html непосредственно перед elem
"afterbegin"    вставить html в начало elem
"beforeend"     вставить html в конец elem
"afterend"      вставить html непосредственно после elem

html
строка, которая будет вставлена именно как HTML
```

Пример:

```
<script>
  div.insertAdjacentHTML('beforebegin', '<p>Привет</p>');
  div.insertAdjacentHTML('afterend', '<p>Пока</p>');
</script>
```

Удаление узлов

```
node.remove()
```

Клонирование узлов

```
elem.cloneNode(true)   клон элемента со всеми атрибутами и дочерними элементами
elem.cloneNode(false)  клон без дочерних элементов
```

Обёртка DocumentFragment

См. в учебнике тут.
Используется редко, потому что вместо него можно вставить массив узлов в другой узел через .append(...array).

Устаревшие методы

Сейчас уже нет причин их использовать, так как современные методы append, prepend, before, after, remove, replaceWith более гибкие и удобные.

```
parentElem.appendChild(node)
добавляет node в конец дочерних элементов parentElem

parentElem.insertBefore(node, nextSibling)
вставляет node перед nextSibling в parentElem

parentElem.replaceChild(node, oldChild)
заменяет oldChild на node среди дочерних элементов parentElem

parentElem.removeChild(node)
удаляет node из parentElem (предполагается, что он родитель node).
```

document.write
записывает html на страницу. Мы можем использовать JavaScript, чтобы создать полноценную веб-страницу и записать её в документ. В современных скриптах он редко встречается из-за следующего важного ограничения: вызов document.write работает только во время загрузки страницы.
Если вызвать его позже, то существующее содержимое документа затрётся.

Стили и классы

```
Классы
elem.className  получить или записать атрибут элемента class="..." в виде строки

elem.classList              специальный объект для добавления/удаления одного класса
add('class1', 'class2')     Добавляет элементу указанные классы
remove('class1', 'class2')  Удаляет у элемента указанные классы
item(Number)                Результат аналогичен вызову classList[index]
length                      кол-во классов у элемента
contains('class1')    Проверяет, есть ли данный класс у элемента (вернет true или false)
toggle('class1' [,Boolean]) Если класс у элемента отсутствует - добавляет, если уже
присутствует - убирает. Если вторым параметром передан false - удаляет указанный класс, а если
true - добавляет.

Стили
elem.style.property         объект, чьи свойства – правила, записанные в атрибуте эл-та
elem.style.color
```

```
elem.style.cssText          переписать или получить весь атрибут style
window.getComputedStyle(el) живой объект со сложенными стилями
```

Существует два способа задания стилей для элемента:
1. Создать класс в CSS файле и присвоить его элементу HTML;
2. Писать стили непосредственно в атрибуте style у HTML элемента: .

JavaScript может:
1. присваивать и удалять классы у элементов;
2. записывать им свойство style.

Запись свойства style используется в случаях, когда что-то надо вычислить динамически. В других случаях предпочтительно использовать навешивание и удаление классов с заранее прописанными стилями (например, если чему-то надо покрасить фон или показать-скрыть на странице)

```
let top = /* сложные расчёты */;
let left = /* сложные расчёты */;

elem.style.left = left; // значение вычисляется во время работы скрипта
elem.style.top = top;   // например, '456px'
```

Работа с классами

```
elem.className  получить или записать атрибут элемента class="…" в виде строки
elem.classList  специальный объект для добавления/удаления одного класса
```

.className

Документация тут.
Отвечает за значение атрибута class="..." элемента. Возвращает или записывает всё, что есть в атрибуте class="...". Т.е. таким образом нельзя дописать класс к существующим, можно только всё перезаписать целиком.
Иными словами, этот метод геттер и сеттер для атрибута class. Он работает с ним как со строкой.

```
let cName = elem.className;
elem.className = 'cName';

elem.className = 'class1 class2 class3'
```

cName – это строка. Если нужно указать несколько классов, они указываются через пробел, как если бы вручную записывались в атрибут.

.classList

Документация тут.
Свойство **classList** возвращает псевдомассив DOMTokenList, содержащий все классы элемента. Это специальный объект с методами для добавления/удаления одного класса.
Результат - псевдомассив DOMTokenList, который можно перебирать в цикле for..of.

```
var elementClasses = elem.classList;
```

ClassList является геттером. Возвращаемый им объект имеет несколько методов:

```
add('class1', 'class1')
Добавляет элементу указанные классы

remove('class1', 'class2')
Удаляет у элемента указанные классы

item(Number)
Результат аналогичен вызову classList[index]

toggle('class1' [,Boolean])
Если класс у элемента отсутствует - добавляет, если уже присутствует - убирает.
Если вторым параметром передан false - удаляет указанный класс, а если true - добавляет.

contains('class1')
Проверяет, есть ли данный класс у элемента (вернет true или false)

length
кол-во классов у элемента
```

Пример:

```
<body class="main page">

// добавление класса
document.body.classList.add('article');

alert(document.body.className);
// main page article
```

Получение и изменение стилей

Работа со стилями

.style

Документация тут.
Используется для получения и установки инлайновых стилей. Это **объект**, который соответствует тому, что написано в атрибуте "style": его свойства – это правила, записанные в атрибуте. Свойство оперирует только значением атрибута "style", без учёта стилей из css-файла и итогового каскада.

Так как возвращается объект, стили прописываются как значения одноимённых свойств этого объекта.

```
elem.style.width="100px"
работает так же, как наличие в атрибуте style строки width:100px
```

Объявленные стили сбрасываются присваиванием значения null или пустой строки

```
elt.style.color = null
elt.style.color = ""
```

Для управления свойствами из нескольких слов используется camelCase.

```
elem.style.backgroundColor    background-color
elem.style.zIndex             z-index
elem.style.borderLeftWidth    border-left-width
```

Стили с браузерным префиксом, например, -moz-border-radius, -webkit-border-radius преобразуются по тому же принципу: дефис означает заглавную букву: MozBorderRadius.

style**.cssText**
Нельзя установить список стилей как, например, div.style="color: red; width: 100px", потому что div.style – это объект, и он доступен только для чтения.
Для задания нескольких стилей в одной строке используется специальное свойство style.cssText.
Это свойство редко используется, потому что такое присваивание удаляет все существующие стили

```
elt.style.cssText = "color: blue; border: 1px solid black";
```

То же самое можно сделать установкой атрибута: div.setAttribute('style', 'color: red...').

window.getComputedStyle

возвращает живой объект (только для чтения), содержащий значения всех CSS-свойств после завершения вычислений значений. Обновляется автоматически когда элемент стилей изменяется.

В отличие от свойства .style, которое тоже возвращает объект, window.getComputedStyle работает с итоговыми стилями элемента, которые складываются как из атрибута style, так и с учётом внешнего файла с css-стилями. Обьект .style следует использовать для установки стилей на специфических элементах.

```
let style = window.getComputedStyle(element [, pseudoElt]);

element
html-элемент, свойства которого необходимо получить

pseudoElt
Строка указывающая на найденный псевдо-элемент. Опускается (или null) для не псевдо-элементов
```

Для правильного получения значения нужно указать точное свойство. Например: paddingLeft, marginTop, borderTopWidth. При обращении к сокращённому: padding, margin, border – правильный результат не гарантируется.

Стили, применяемые к посещённым :visited ссылкам, скрываются!

Размеры и прокрутка элементов

```
Размеры и прокрутка
.offsetParent    ссылка на предка элемента
.offsetTop       расстояние вверх до родителя
.offsetLeft      расстояние слева от родителя

.offsetWidth     ширина + padding + border
.offsetHeight    высота + padding + border

.clientTop       толщина рамки border (чаще всего)
.clientLeft

.clientWidth     ширина + padding
.clientHeight    высота + paddind

.scrollWidth     вся ширина элемента на странице, в т.ч. с невидимой частью
.scrollHeight    вся высота элемента на странице, в т.ч. с невидимой частью

.scrollLeft      ширина невидимой, уже прокрученной части элемента (можно записывать)
.scrollTop       высота невидимой, уже прокрученной части элемента (можно записывать)


let scrollBottom = elem.scrollHeight - elem.scrollTop - elem.clientHeight;
узнать размер прокрутки снизу
```

Как получить размер прокрутки снизу?
вся высота - прокрученная сверху часть - видимая часть. Границы не считать!

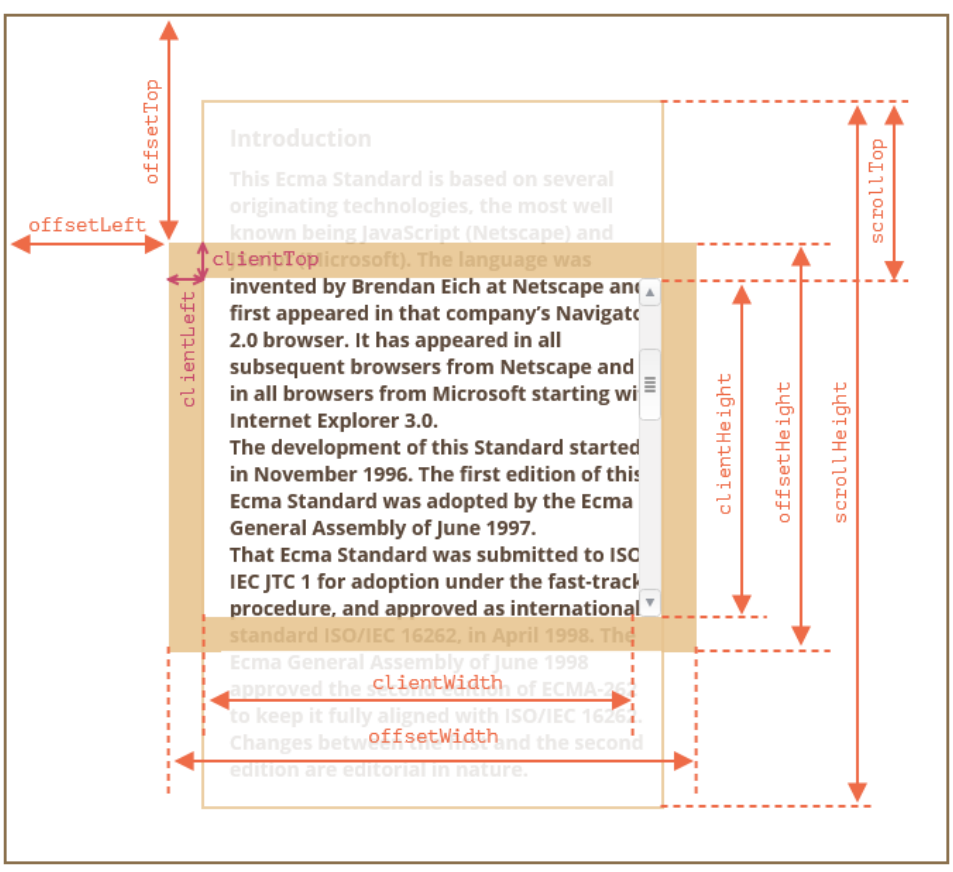
```
let scrollBottom = elem.scrollHeight - elem.scrollTop - elem.clientHeight;
```

Как узнать ширину полосы прокрутки окна?
Размеры с границей – область внутри рамок.
Похоже, что в решении ошибка, потому что должны минусоваться границы border.

```
let scrollWidth = div.offsetWidth - div.clientWidth;
```

Если элементы не отображены на странице, их метрики равны нулю.
Координаты и размеры в JavaScript устанавливаются только для видимых элементов.
Если элемент (или любой его родитель) имеет display:none или отсутствует в документе, то все его метрики равны нулю (или null, если это offsetParent).
Можно использовать это, чтобы делать проверку на видимость:

```
function isHidden(elem) {
  return !elem.offsetWidth && !elem.offsetHeight;
}
```



Внешние отступы

.offsetParent

Документация тут.
Возвращает ссылку на ближайшего предка элемента (в html-иерархии).

Возвращает null, если:
- элемент или его родитель установлен в display: none
- у элемента position: fixed (firefox вернёт body)
- элемент – это body или html.

offsetTop и offsetLeft отсчитываются от края этого предка.

```
parentObj = element.offsetParent;
```

.offsetTop

Документация тут.
Возвращает расстояние текущего элемента по отношению к родителю. Это количество пикселей на которые делается отступ сверху, отсносительно родительского элемента.
Расстояние вычисляется от края границы border родительского элемента до края границы border элемента.

```
topPos = element.offsetTop;
```

Пример использования:

```
var div = document.getElementById("div1");
var topPos = div.offsetTop;

if (topPos > 10) {
  // если объект имеет отступ больше 10 пикселей от своего родителя
}
```

.offsetLeft

Документация тут.
содержит левое смещение (до границы элемента) элемента относительно offsetParent.

```
left = element.offsetLeft;
```

Размеры с границей

Эти свойства – внешняя высота и ширина элемента с учётом border.
Формула: css ширина + (padding + border) * 2

.offsetWidth

Документация тут.
Возвращает ширину элемента. Как правило, offsetWidth — это значение, включающее горизонтальный отступ элемента, ширину вертикального скроллбара (если он есть) и CSS ширину.

```
var offsetWidth = element.offsetWidth;
```

.offsetHeight

Документация тут.
Высота элемента с учетом вертикальных полей и границ в пикселях. Свойство неизменяемое, только для чтения. Возвращаемое значение - целочисленное.

```
let offsetHeight = element.offsetHeight;
```

Ширина границы

Чаще всего, это размеры границ border элемента, левой и верхней.
Если говорить точно, это отступы внутренней части элемента от внешней. Она возникает, когда документ располагается справа налево (операционная система на арабском языке или иврите). Полоса прокрутки в этом случае находится слева, и тогда свойство clientLeft включает в себя ещё и ширину полосы прокрутки.

.clientTop

Документация тут.
Толщина верхней границы элемента в пикселях. Не включает в себя margin и padding. Свойство только для чтения.

```
var top = element.clientTop;
```

.clientLeft

Документация тут.
Ширина левой границы элемента в пикселях. Не включает в себя padding и margin.

```
var left = element.clientLeft;
```

Область внутри рамок

Эти свойства – размер области внутри рамок элемента.
Они включают в себя ширину области содержимого вместе с внутренними отступами padding, но без прокрутки.
Если нет внутренних отступов padding, то clientWidth и clientHeight будут считаться вместе с полосой прокрутки.
Иными словами, это видимая часть содержимого (с оговоркой про отсутствующий padding).

.clientWidth

Документация тут.
Включает: ширину содержимого, padding.
Не включает: border, полосу прокрутки.
Свойство равно 0 для инлайн элементов и элементов без CSS.

```
var intElemClientWidth = element.clientWidth;
```

.clientHeight

Документация тут.
Включает: высоту содержимого, padding.
Не включает: border, ролосу прокрутки.
Для элементов без CSS-стилей, или элементов каркаса строчной разметки - значение равно нулю.
Может быть вычислено по формуле CSS height + CSS padding - высота горизонтального скролла.

Вся высота с прокруткой

Эти свойства – как clientWidth и clientHeight, но также включают в себя прокрученную (которую не видно за границами экрана) часть элемента. Иными словами, это вся высота содержимого.
Эти свойства можно использовать, чтобы «распахнуть» элемент на всю ширину и высоту:

```
element.style.height = `${element.scrollHeight}px`;
```

.scrollHeight

.scrollWidth

Невидимая часть

Ширина и высота невидимой, прокрученной в данный момент, части элемента слева и сверху. Т.е. это та часть, которую уже не видно, она за пределами экрана.
Иными словами, это уже прокрученная часть.

В отличие от большинства свойств, которые доступны только для чтения, значения scrollLeft/scrollTop можно изменять, и браузер выполнит прокрутку элемента.

.scrollLeft

Документация тут.
Свойство получает или устанавливает количество пикселей, на которое контент элемента прокручен влево.

```
// Получаем количество прокрученных пикселей
var sLeft = element.scrollLeft;

// Устанавливаем количество прокрученных пикселей
element.scrollLeft = 10;
```

```
element.scrollTop += 10
```

.scrollTop

Документация тут.
Свойство считывает или устанавливает количество пикселей, прокрученных от верха элемента. scrollTop измеряет дистанцию от верха элемента до верхней точки видимого контента. Когда контент элемента не создаёт вертикальную прокрутку, его scrollTop равно 0.

```
// Получаем количество прокрученных пикселей
var  intElemScrollTop = someElement.scrollTop;

// Устанавлием количество прокрученных пикселей
element.scrollTop = intValue;
```

Не стоит брать width и height из CSS

CSS-высоту и ширину можно извлечь, используя getComputedStyle. Но так делать нельзя. На то есть две причины:

Во-первых, CSS-свойства width/height зависят от другого свойства – box-sizing. Получается, что изменение box-sizing, к примеру, для более удобной вёрстки, сломает такой JavaScript.
Во-вторых, в CSS свойства width/height могут быть равны auto, например, для инлайнового элемента.

Есть и ещё одна причина: полоса прокрутки. Бывает, без полосы прокрутки код работает прекрасно, но стоит ей появиться, как начинают проявляться баги. Так происходит потому, что полоса прокрутки «отъедает» место от области внутреннего содержимого в некоторых браузерах. Таким образом, реальная ширина содержимого меньше CSS-ширины. Как раз это и учитывают свойства clientWidth/clientHeight.

В чём отличие между

```
getComputedStyle(elem).width
elem.clientWidth
```

1. clientWidth возвращает число, а getComputedStyle(elem).width – строку с px на конце.
2. getComputedStyle не всегда даст ширину, он может вернуть, к примеру, "auto" для строчного элемента.
3. clientWidth соответствует внутренней области элемента, включая внутренние отступы padding, а CSS-ширина (при стандартном значении box-sizing) соответствует внутренней области без внутренних отступов padding.
4. Если есть полоса прокрутки, и для неё зарезервировано место, то некоторые браузеры вычитают его из CSS-ширины (т.к. оно больше недоступно для содержимого), а некоторые – нет. Свойство clientWidth всегда ведёт себя одинаково: оно всегда обозначает размер за вычетом прокрутки, т.е. реально доступный для содержимого.

Размеры и прокрутка окна

```
Размеры и прокрутка окна
Размеры окна
document.documentElement.clientWidth   ширина окна браузера (без полосы прокрутки)
document.documentElement.clientHeight  высота окна браузера (без полосы прокрутки)

window.innerWidth                  ширина окна браузера (включая полосу прокрутки)
window.innerHeight                 высота окна браузера (включая полосу прокрутки)

Прокрутка окна и элементов
window.pageYOffset                 показать текущую прокрутку по вертикали (только чтение)
window.pageXOffset                 показать текущую прокрутку по горизонтали (можно без window)

documentElement.scrollLeft         узнать прокрутку или установить её (окно и элементы)
documentElement.scrollTop          узнать прокрутку или установить её (окно и элементы)

window.scrollBy(x, y)              прокрутка относительно текущего положения на n пиксесей
window.scrollTo(pageX, pageY)      прокрутка на абсолютные координаты

elem.scrollIntoView(true)          прокрутить окно к elem (появится в верху)
elem.scrollIntoView(false)         прокрутить окно к elem (появится внизу)

document.documentElement.scrollTop старые способы прокрутить окно
document.documentElement.scrollLeft
window.scrollByLines
window.scrollByPages

document.body.style.overflow = "hidden"   отключить прокрутку
document.body.style.overflow = ""         включить обратно
```

Размеры окна

Эти свойства показывают только доступную ширину и высоту окна, потому что вычитают полосу прокрутки. Предпочтительнее использовать их.

```
document.documentElement.clientWidth
document.documentElement.clientHeight
```

Эти свойства показывают ширину и высоту окна, включая элемент прокрутки.

```
window.innerWidth
window.innerHeight
```

Размеры документа

Корневым элементом документа является documentElement. Тем не менее, его свойства .scrollWidth и .scrollHeight не всегда возвращают корректный размер документа. Их размер может быть меньше, чем .clientHeight, а с точки зрения элемента это невозможная ситуация. Как всегда, это несоответствие историческое.

Чтобы надёжно получить высоту документа, следует взять максимальное из этих свойств:

```
let height = Math.max(
  document.body.scrollHeight, document.documentElement.scrollHeight,
  document.body.offsetHeight, document.documentElement.offsetHeight,
  document.body.clientHeight, document.documentElement.clientHeight
);
```

Текущая прокрутка

Чаще используются эти свойства (только для чтения):

```
window.pageYOffset
window.pageXOffset
```

Поскольку window – это глобальный объект, то можно писать просто: pageYOffset

Эти свойства тоже работают для большинства браузеров:

```
documentElement.scrollLeft
documentElement.scrollTop
document.body.scrollLeft для старых Safari
```

Как прокручивать

Для прокрутки страницы DOM должен быть полностью построен.
Если попытаться прокрутить страницу из <head>, это не сработает.

Специальные методы для прокрутки страницы (лучше использовать их)

```
window.scrollBy(x, y)           прокрутка относительно текущего положения на n пиклесей
window.scrollTo(pageX, pageY)   прокрутка на абсолютные координаты
```

См. также window.scrollByLines, window.scrollByPages

e.scrollLeft

e.scrollTop

Обычные элементы можно прокручивать, изменяя .scrollTop и .scrollLeft. Эти же свойства можно применять для прокрутки страницы (с оговоркой про Safari):

```
Старые способы
document.documentElement.scrollTop
document.documentElement.scrollLeft
```

e.scrollIntoView(top)

Документация тут.
Прокручивает страницу, чтобы elem оказался вверху или внизу, т.е. он будет совмещён либо с верхней, либо с нижней частью страницы.

```
elem.scrollIntoView(true)
elem появится в верху

elem.scrollIntoView(false)
elem появится внизу
```

window.scrollBy(x, y)

Документация тут.
Прокручивает документ на указанные величины.
Положительные значения приведут к прокрутке страницы вправо и вниз. Отрицательные – влево и вверх.

```
window.scrollBy(x, y)

// Прокрутка на один экран вертикально вниз.
window.scrollBy(0, window.innerHeight);
```

window.scrollTo(x-coord, y-coord)

Документация тут.
Прокрутка документа до указанных координат.

```
window.scrollTo(x-coord, y-coord)
window.scrollTo(options)

options
Объект с тремя возможными параметрами:
{
  top,
  left,
  'behavior' (строка smooth, instant, либо auto)
}
```

Отключить прокрутку

Иногда нам нужно сделать документ «непрокручиваемым». Например, при показе большого диалогового окна над документом – чтобы посетитель мог прокручивать это окно, но не документ.

Делается это так:

```
document.body.style.overflow = "hidden"   отключить прокрутку
document.body.style.overflow = ""         включить обратно
```

Недостатком этого способа является то, что сама полоса прокрутки исчезает и содержимое страницы расширится, текст «прыгнет», заняв освободившееся место. Это можно выличить навешиванием padding в соответствующем размере.

Координаты

```
Координаты
Относительно окна
let el = document.elementFromPoint(x, y)
вернуть элемент, который находится по этим координатам


domRect = el.getBoundingClientRect()
domRect.x        X-координата начала прямоугольника относительно окна
domRect.y        Y-координата начала прямоугольника относительно окна

domRect.width    ширина прямоугольника (м.б. отрицательным)
domRect.height   высота прямоугольника (м.б. отрицательным)

domRect.top      Y-координата верхней границы прямоугольника (доп. зависимое свойство)
domRect.bottom   Y-координата нижней границы прямоугольника (доп. зависимое свойство)

domRect.left     X-координата левой границы прямоугольника (доп. зависимое свойство)
domRect.right    X-координата правой границы прямоугольника (доп. зависимое свойство)


Относительно документа надо вычислять
pageY = clientY + высота вертикально прокрученной части документа
pageX = clientX + ширина горизонтально прокрученной части документа

Функция вычисления координат объекта относительно документа
function getCoords(elem) {
  let box = elem.getBoundingClientRect();
  return {
    top: box.top + pageYOffset,
    left: box.left + pageXOffset
  };
}
```

Большинство JS метоов работают с координатами относительно:
1. окна браузера
Например, position: fixed отсчитывается от верхнего левого края окна.
Далее обозначаются как clientX, clientY

2. документа
Например, position: absolute.
Далее обозначаются как pageX, pageY

Координаты в контексте окна подходят для использования с position:fixed,

Координаты относительно документа – для использования с position:absolute.

Относительно окна

.getBoundingClientRect()

Метод возвращает объект DOMRect.
С помошью этого объекта можно узнать размер элемента и его позицию относительно окна браузера (viewport).
Координаты и размеры прямоугольника – дробное число. Мб отрицательное, если элемент прокручен за пределы окна браузера.

```
domRect = element.getBoundingClientRect();
```

Свойства объекта:

```
координаты начала прямоугольника относительно окна
domRect.x
domRect.y

ширина и высота прямоугольника (могут быть отрицательными)
domRect.width
domRect.height

дополнительные, зависимые свойства
Y-координата верхней или нижней границы прямоугольника
domRect.top
domRect.bottom

X-координата левой или правой границы прямоугольника
domRect.left
domRect.right
```

Top, left и всё остальное будет меняться в зависимости от прокрутки окна.
Internet Explorer и Edge не поддерживают свойства x/y

document.**elementFromPoint**(x, y)

Документация тут.
Возвращает элемент, который находится по указанным координатам окна браузера и является самым верхним.

Метод работает, только если координаты относятся к видимой части содержимого окна.
Если любая из координат представляет собой отрицательное число или превышает размеры окна, то возвращается null.

```
const element = document.elementFromPoint(x, y)
```

Применение для fixed позиционирования

Чаще всего нам нужны координаты для позиционирования чего-либо.
Чтобы показать что-то около нужного элемента, мы можем вызвать getBoundingClientRect, чтобы получить его координаты элемента, а затем использовать CSS-свойство position вместе с left/top (или right/bottom).

Но обратите внимание на одну важную деталь: при прокрутке страницы сообщение уплывает от кнопки.
Причина весьма очевидна: сообщение позиционируется с помощью position:fixed, поэтому оно остаётся всегда на том же самом месте в окне при прокрутке страницы.
Чтобы изменить это, нам нужно использовать другую систему координат, где сообщение позиционировалось бы относительно документа, и свойство position:absolute.

Относительно документа

Не существует стандартного метода, который возвращал бы координаты элемента относительно документа, но такие координаты можно вычислить.

```
pageY = clientY + высота вертикально прокрученной части документа
pageX = clientX + ширина горизонтально прокрученной части документа


.getBoundingClientRect()
Возвращает объект с прямоугольником, в который заключён элемент

pageYOffset
pageXOffset
Узнать текущую прокрутку окна браузера


Функция вычисления координат объекта относительно документа
function getCoords(elem) {
  let box = elem.getBoundingClientRect();

  return {
    top: box.top + pageYOffset,
    left: box.left + pageXOffset
  };
}
```

Функция показывает координаты клика (из задачи)

```
document.onclick = function(e) {
  coords.innerHTML = e.clientX + ':' + e.clientY;
};
```

clientX – это события мыши. Документация тут.
Перенести потом в соответствующий раздел.
</>

Введение в события

Event кратко

```
e.addEventListener('event', handler, {options});
e.removeEventListener("click", foo);

event.target         ссылка на объект, который был инициатором события
event.currentTarget  ссылка на элемент, в котором в данный момент обрабатывается событие (this)

event.stopPropagation()              выполняет обработку и останавливает всплытие события
event.stopImmediatePropagation();    останавливает и текущую обработку события, и всплытие

event.preventDefault()  отменить действие браузера по умолчанию в eventListener
event.defaultPrevented  будет true, если действие по умолчанию было отменено

event.isTrusted              true, если событие сгенерировано браузером, а не из кода
target.dispatchEvent(event)  запустить событие через код


new Event( typeArg, PeventInit {
  "bubbles": false,
  "cancelable": false,
  "composed": false
});

new CustomEvent
```

Введение в браузерные события

```
Чтобы код выполнился только после полной загрузки документа, надо писать его внутри этой
конструкции:
document.addEventListener("DOMContentLoaded", function() { код });

<e onclick="alert('Клик!')">   повесить обработчик через html
e.onclick = sayThanks;         повесить обрабочик через dom-свойство
e.onclick = null               удалить обработчик

e.addEventListener("click", foo);     лучший способ вешать обработчики
e.removeEventListener("click", foo);  удалить обработчик
```

События

Событие – это сигнал от браузера о том, что что-то произошло. Все DOM-узлы подают такие сигналы.
Некоторые часто используемые DOM-события:

Перечень событий в DOM
https://developer.mozilla.org/ru/docs/Web/Events

События мыши:

click	клик на элементе левой кнопкой, или тап на сенсорном экране
contextmenu	клик правой кнопкой мыши
mouseover	мышь наведена на элемент
mouseout	мышь покидает элемент
mousedown	кнопка мыши нажата на элементе
mouseup	кнопка мыши отжата на элементе
mousemove	при движении мыши

Клавиатурные события:

keydown	зажатие клавиши
keyup	отпуск клавиши

События на элементах управления:

submit	пользователь отправил форму <form>.
focus	пользователь фокусируется на элементе, например нажимает на <input>.

CSS events:

transitionend	когда CSS-анимация завершена.

События документа:
DOMContentLoaded
когда HTML загружен и обработан, DOM документа полностью построен и доступен. Вешается только через EventListener.

```
document.addEventListener("DOMContentLoaded", function() {
  весь JS код пишется внутри этой функции
});
```

Обработчики событий

Обработчик события – функция, которая сработает, как только событие произошло.

Обычно события обрабатываются асинхронно. То есть, если браузер обрабатывает onclick и в процессе этого произойдёт новое событие, то оно ждёт, пока закончится обработка onclick.
Смотри «Вложенные события обрабатываются синхронно».
Исключением является ситуация, когда событие инициировано из обработчика другого события.
(#? Неточность какая-то)

Назначить обработчик события можно несколькими способами:
1. в атрибуте HTML
2. в свойстве DOM-объекта
3. через объект-обработчик addEventListener

- Внутри обработчика события **this** ссылается на текущий элемент, то есть на тот, на котором висит (т.е. назначен) обработчик.
- Обработчик в любом случае будет храниться в dom-свойстве объекта, каким бы образом он не задавался.
- Убрать обработчик можно назначением `elem.onclick = null`

setAttribute для обработчиков не работает.
Не используйте setAttribute для обработчиков.Такой вызов работать не будет:

```
// при нажатии на body будут ошибки,
// атрибуты всегда строки, и функция станет строкой
document.body.setAttribute('onclick', function() { alert(1) });
```

Через атрибут HTML
Атрибут называется on + событие.
Важно не забыть поставить вызов() функции, если в атрибут передаётся готовая функция.
В примерах ниже, при клике на элементе выполнится код, указанный в атрибуте.

```
<input value="Нажми меня" onclick="alert('Клик!')" type="button">

// функция makeRed() объявлена где-то выше
<div onclick="makeRed()">abc</div>

// не функция, а строчка кода
<div onclick="this.style.backgroundColor = 'red'">abc</div>
```

Через свойство DOM-объекта
Свойство называется .on + событие. Важно, что вызов функции() в этом случае не ставится:

```
// тело функции
elem.onclick = function() {
  alert('Спасибо');
};

// без вызова()
elem.onclick = sayThanks;

// удалить обработчик
elem.onclick = null
```

addEventListener

В отличие от других способов, через addEventListener можно вешать сразу несколько обработчиков.
На некоторые события можно повесить обработчик только через EventListener.
Важно! Имена событий прописываются без on. Например, 'click', а не 'onclick'.

Обработчики, назначенные через addEventListener, не мешают обработчикам, назначенным через html или dom-свойство (работают одновременно оба).

Документация тут.

Добавить обработчик:

```
element.addEventListener('event', handler [, options]);
element.removeEventListener('event', handler[, options]);

event
Имя события без on, например "click"
handler
Ссылка на функцию-обработчик

Options {           объект со свойствами
  once: true,       обработчик автоматом удалится после выполнения
  capture: false,   фаза, на которой сработает обработчик. Если true – сработает на погружении.
  passive: true     обработчик никогда не вызовет preventDefault()
}
```

.addEventListener позволяет навешивать несколько обработчиков и может использоваться с обработчиком, который повесили из dom:

```
elem.onclick = () => alert("Привет");
elem.addEventListener("click", handler1);
elem.addEventListener("click", handler2);
```

Удалить обработчик:

```
element.removeEventListener(event, handler[, options]);
```

Для удаления нужно передать именно ту функцию-обработчик которая была назначена. Это значит, что функция должна быть объявлена где-то ещё, а в обрабочтик передана по ссылке. Если в remove просто переписать код функции, то JS будет считать это созданием новой функции.
Если функцию обработчик не сохранить в другом месте, мы не сможем её удалить.

#? Что такое options?
Чтобы убрать обработчик removeEventListener, нужно указать ту же функцию и ту же фазу.
Если мы добавили обработчик вот так addEventListener(..., true), то мы должны передать то же значение аргумента capture в removeEventListener(..., true), когда снимаем обработчик.

handleEvent

Непонятная тема, документация тут и в ней про что-то другое.
В уроке говорится, что в качестве обработчика можно передавать не только функцию, но и объект, у которого есть метод handleEvent.

В этом примере в качетсве обработчика создаётся пустой **объект** { }, и ему присваивается метод handleEvent, который будет обрабатывать событие:

```
elem.addEventListener('click', {
  handleEvent(event) {
    alert(event.type + " на " + event.currentTarget);
  }
});
```

Также, можно создать класс, который будет создавать такие объекты с обработчиком и передавать эти объекты в качестве обработчика:

```
class MyClass {
  handleEvent(event) {
    switch(event.type) {
      case 'mousedown':
        elem.innerHTML = "Нажата кнопка мыши";
        break;
      case 'mouseup':
        elem.innerHTML += "...и отжата.";
        break;
    }
  }
}

let menu = new MyClass();

elem.addEventListener('mousedown', menu);
elem.addEventListener('mouseup', menu);
```

Здесь один и тот же объект обрабатывает оба события. Нужно явно назначить оба обработчика через addEventListener.

Метод handleEvent не обязательно должен выполнять всю работу сам. Он может вызывать другие методы, которые заточены под обработку конкретных типов событий. Это будет правильно, потому что это упростит поддержку кода:

```
<button id="elem">Нажми меня</button>

<script>

  class MyClass {
    handleEvent(event) {
      // mousedown -> onMousedown
      let method = 'on' + event.type[0].toUpperCase() + event.type.slice(1);
      this[method](event);
    }

    onMousedown() {
      elem.innerHTML = "Кнопка мыши нажата";
    }

    onMouseup() {
      elem.innerHTML += "...и отжата.";
    }
  }

  let menu = new MyClass();
  elem.addEventListener('mousedown', menu);
  elem.addEventListener('mouseup', menu);

</script>
```

Event: объект события

Документация по объекту event тут.
Сами события (не объекты, а именно события) здесь: Element => events.
Ещё события: Pointer Events
От event наследуют другие объекты, например MouseEvent.
DOM Events описывает 3 фазы прохода события (погружение, текущий, всплытие)

Когда происходит событие, браузер создаёт объект события, записывает в него различную информацию и передаёт его в качестве аргумента функции-обработчику.

```
element.addEventListener('click', function(event) {
  event.type;   // click
```

```
});
```

События

Сами события (не объекты, а именно события) здесь: Element => events.
Ещё события: Pointer Events

```

```

Event объект

Список **свойств** Event тут: **Свойства**
Список **методов** Event тут: **Методы**

Некоторые свойства и методы:

```
Event()
Создаёт объект Event и возвращает его вызывающему.

Свойства
Event.type
Название события.

Event.target
Ссылка на целевой объект, на котором произошло событие.
Event.currentTarget
Ссылка на текущий зарегистрированный объект, на котором обрабатывается событие. Это объект,
которому планируется отправка события; поведение можно изменить с использованием
перенаправления (retargeting).

Event.bubbles
Логическое значение, указывающее, всплыло ли событие вверх по DOM или нет.
Event.defaultPrevented
Показывает, была ли для события вызвана функция event.preventDefault().

Event.eventPhase
Указывает фазу процесса обработки события.
Event.timeStamp
Время, когда событие было создано (в миллисекундах).

Event.isTrusted
Показывает было или нет событие инициировано браузером (например, по клику мышью) или из
скрипта (например, через функцию создания события, такую как event.initEvent)
Event.deepPath
Массив DOM-узлов, через которые всплывало событие.


Методы
Event.preventDefault()
Отменяет действие браузера по умолчанию.
Event.stopImmediatePropagation()
Для конкретного события не будет больше вызвано обработчиков. Дальнейшее всплытие (и
погружение) останавливается.
Event.stopPropagation()
Остановка распространения события далее по DOM.
```

MouseEvent

Некоторые свойства и методы:

```
MouseEvent.altKey
true, если клавиша alt была нажата во время движения мыши.
MouseEvent.shiftKey
true если клавиша shift была нажата, когда произошло событие мыши.
MouseEvent.ctrlKey Только для чтения
true, если клавиша control была нажата во время движения мыши.
MouseEvent.metaKey
true, если клавиша meta была нажата во время движения мыши.

MouseEvent.button
Представляет код клавиши, нажатой в то время, когда произошло событие мыши.
MouseEvent.buttons
Отображает, какие клавиши были нажаты во время движения мыши.

MouseEvent.clientX
Отображение X координат курсора мыши в локальной системе координат (DOM контент).
MouseEvent.clientY
Отображение Y координат курсора мыши в локальной системе координат (DOM контент).

MouseEvent.offsetX
Отображает X координат указателя мыши относительно позиции границы отступа целевого узла.
MouseEvent.offsetY
Отображает Y координат указателя мыши относительно позиции границы отступа целевого узла.


MouseEvent.relatedTarget
Второстепенная цель события, если таковая есть.

MouseEvent.pageX
Отображает X координат указателя мыши относительно всего документа.
MouseEvent.pageY
Отображает Y координат указателя мыши относительно всего документа.
MouseEvent.screenX
Отображает X координат указателя мыши в пространстве экрана.
MouseEvent.screenY
Отображает Y координат указателя мыши в пространстве экрана.

MouseEvent.which
Возвращает код последней нажатой клавиши, когда произошло событие мыши.
```

Всплытие и погружение

```
Всплытие и погружение

event.target         ссылка на объект, который был инициатором события
event.currentTarget  ссылка на элемент, в котором в данный момент обрабатывается событие (this)

event.stopPropagation()              выполняет обработку и останавливает всплытие события
event.stopImmediatePropagation();    останавливает и текущую обработку события, и всплытие
```

Когда на элементе происходит событие, обработчики сначала срабатывают на нём, потом на его родителе и так далее вверх по цепочке предков.

Например, есть 3 вложенных элемента FORM > DIV > P с обработчиком на каждом:

```
<form onclick="alert('form')">FORM
  <div onclick="alert('div')">DIV
    <p onclick="alert('p')">P</p>
  </div>
</form>
```

Клик по внутреннему <p> вызовет обработчик onclick:

1. Сначала на самом <p>.
2. Потом на внешнем .
3. Затем на внешнем <form>.
4. И так далее вверх по цепочке до самого document.

В итоге, если кликнуть на <p>, то мы увидим три оповещения: p → div → form.
Этот процесс называется «всплытием», потому что события «всплывают» от внутреннего элемента вверх через родителей.

Если у нас несколько обработчиков одного события, назначенных addEventListener на один элемент, в рамках одной фазы, то их порядок срабатывания – тот же, в котором они установлены

У объекта события есть несколько свойств, которые нацелены на работу со всплытием.

event.target

Элемент, на котором изначально произошло событие. Это именно инициатор события, поэтому он никогда не изменяется. Документация тут.

event.currentTarget

Элемент, в котором событие находится (обрабатывается) в данный момент. Эквивалент this.
(документация тут)

Отличия Event.target от this и Event.currentTarget:
1. event.target – это «целевой» элемент, на котором произошло событие, в процессе всплытия он неизменен.
2. this – это «текущий» элемент, до которого дошло всплытие, на нём сейчас выполняется обработчик.

event.stopPropagation()

Препятствует продвижению события дальше, но на текущем элементе все обработчики будут вызваны.
Документация тут.
Любой промежуточный обработчик может остановить всплытие через event.stopPropagation().
Важно, что текущая обработка события (которая повешена на элемент) выполнится. Т.е. элемент запустит все свои обработчики, но не передаст событие наверх.

```
event.stopPropagation();

Пример
<body onclick="alert(`сюда всплытие не дойдёт`)">
  <button onclick="event.stopPropagation()">Кликни меня</button>
</body>
```

event.stopImmediatePropagation()

Предотвращает и всплытие, и обработку уже на текущем элементе.
Документация тут.

```
event.stopImmediatePropagation();
```

Погружение

Существует ещё одна фаза из жизненного цикла события – «погружение» (иногда её называют «перехват»).

Стандарт DOM Events описывает 3 фазы прохода события:
1. Фаза погружения (capturing phase) – событие сначала идёт сверху вниз.
2. Фаза цели (target phase) – событие достигло целевого(исходного) элемента.
3. Фаза всплытия (bubbling stage) – событие начинает всплывать.

Иными словами, технически событие сначала идёт вниз от Window до целевого объекта, потом вновь возвращается в Window.
Поймать погружение можно только через addEventListener с третьим аргументом: ('event', handler, {capture: true})
Любые другие способы навешивания обработчиков (через dom, HTML и addEventListener(event, handler) с двумя аргументами) работают только на 2-ой и 3-ей фазах.

Чтобы поймать погружение, нужно использовать .addEventListener с тетьим аргументом – объектом capture со значениями true или false. Можно просто прописать логическое значение кратко:

```
elem.addEventListener('click', foo, {capture: true})
elem.addEventListener('click', foo, true)

false  событие будет перехвачено при всплытии (по умолчанию)
true   событие будет перехвачено при погружении
```

Чтобы убрать обработчик removeEventListener, нужно указать ту же функцию и ту же фазу.
Если мы добавили обработчик вот так addEventListener(..., true), то мы должны передать то же значение аргумента capture в removeEventListener(..., true), когда снимаем обработчик.

event.eventPhase

Свойство содержит номер фазы, на которой событие было поймано.
Документация тут.
Возвращает целое число, соответствующее одной из 4 констант:

```
let phase = event.eventPhase;

0   none
1   capturing (погружение)
2   at tagret
3   bubbling (всплытие)
```

Делегирование событий

В случае, когда надо однотипно обработать несколько дочерних элементов, можно делегировать их обработку родительскому элементу, а не вешать обработчик на каждый дочерний.
Особенности подхода можно рассмотреть на примере.

Структура страницы:

```
        -> child -> text
parent  -> child -> text
        -> child -> text
```

При клике на child или любой его дочерний элемент, событие должно делегироватья в parent и в нём перекрасить фон для child.

Код:

```
let parent = document.body.querySelector('.parent');
let previousChild;

function changeColor(ev) {
  let target = ev.target;                  // 1
  let child = target.closest('.child');    // 2

  if (!child) return;                      // 3
  if (!parent.contains(child)) return;     // 4
```

```
  child.classList.toggle('highlight');
}

parent.addEventListener('click', changeColor);
```

1. Определить инициатор события. Это может быть как сам child, так и его дочерние элементы.
2. Найти элемент-предок, которому надо покрасить фон. Нельзя просто покрасить инициатора, потому что если клик произошёл на вложенном элементе text, то он будет инициатором и фон покрасится не у child, а у text. Это делется через проверку .closest()
3, 4. Если клик произошёл на каком-то «левом» элементе, который вложен в parent, но при этом не вложен в child, то он не пройдёт проверку, вернёт false и ничего не покрасится.

HTML и CSS страницы:

```
<div class="parent">

    <div class="child">
        <div class="text">first</div>
    </div>

    <div class="child">
        <div class="text">second</div>
    </div>

    <div class="child">
        <div class="text">third</div>
    </div>

</div>
```

```
.child {
  margin: 10px;
  width: 100px;
  height: 100px;
  background-color: paleturquoise;
  display: flex;
  justify-content: center;
  align-items: center;
}

.text {
  padding: 5px 0;
  width: 80%;
  text-align: center;
  background-color: pink;
}

.highlight {
  background-color: #ff92f6;
}
```

Применение делегирования: действия в разметке

Есть объект с методами save, load, search. Есть соответствующие кнопки.
Надо сделать один обработчик для всего меню, а кнопкам добавить атрибуты «data-action».

```
<div id="menu">
  <button data-action="save">Сохранить</button>
  <button data-action="load">Загрузить</button>
  <button data-action="search">Поиск</button>
</div>


class Menu {
  constructor(elem) {
    this._elem = elem;
    elem.onclick = this.onClick.bind(this); // (*)
  }

  save() {
    alert('сохраняю');
  }

  load() {
    alert('загружаю');
  }

  search() {
    alert('ищу');
  }

  onClick(event) {
    let action = event.target.dataset.action;
    if (action) {
      this[action]();
    }
  };
}

new Menu(menu);
```

```
<div id="menu">
  <button data-action="save">Сохранить</button>
  <button data-action="load">Загрузить</button>
  <button data-action="search">Поиск</button>
</div>
```

Приём проектирования «поведение»

Приём проектирования «поведение» состоит из двух частей:
1. Элементу ставится пользовательский атрибут, описывающий его поведение.
2. При помощи делегирования ставится обработчик на документ, который ловит все клики (или другие события) и, если элемент имеет нужный атрибут, производит соответствующее действие.

Поведение: «Счётчик»
На странице есть 2 кнопки. При клике на кнопку, их значение (value) должно увеличиваться.
Для этого этим кнопкам добавляется атрибут «data-counter». При клике на такую кнопку, их значение будет +=1:

```
HTML
Счётчик: <input type="button" value="1" data-counter>
Ещё счётчик: <input type="button" value="2" data-counter>

JS
<script>
  document.addEventListener('click', function(event) {

    if (event.target.dataset.counter != undefined) { // если есть атрибут...
      event.target.value++;
    }

  });
</script>
```

Элементов с атрибутом data-counter может быть сколько угодно. Новые могут добавляться в HTML-код в любой момент. При помощи делегирования мы фактически добавили новый «псевдостандартный» атрибут в HTML, который добавляет элементу новую возможность («поведение»).

Поведение: «Переключатель» (Toggler)
при клике на элемент с атрибутом «data-toggle-id» будет скрываться/показываться элемент с заданным id:

```
<button data-toggle-id="subscribe-mail">
  Показать форму подписки
</button>

<form id="subscribe-mail" hidden>
  Ваша почта: <input type="email">
</form>


<script>
  document.addEventListener('click', function(event) {
    let id = event.target.dataset.toggleId;
    if (!id) return;

    let elem = document.getElementById(id);

    elem.hidden = !elem.hidden;
  });
</script>
```

Теперь для того, чтобы добавить скрытие-раскрытие любому элементу, даже не надо знать JavaScript, можно просто написать атрибут data-toggle-id.
Не нужно писать JavaScript-код для каждого элемента, который должен так себя вести, просто используем поведение. Обработчики на уровне документа сделают это возможным для элемента в любом месте страницы.

Действия браузера по умолчанию

```
Действия браузера по умолчанию
event.preventDefault()  отмена действия браузера в eventListener
onclick="return false"  отмена действия браузера, если событие назначено через 'on'

event.defaultPrevented  будет равняться true, если действие по умолчанию было отменено
```

Пример действий по умолчанию:
- клик по ссылке – переход на новый URL.
- нажатие на кнопку «отправить» в форме – отсылка на сервер.
- зажатие кнопки над текстом и движение – выделение текста.

Отмена действий браузера

Есть два способа отменить действие по умолчанию:
1. event.preventDefault(), если обработчик вешается через addEventListener
2. вернуть из функции false, если обработчик назначен через **on**<событие> (не через addEventListener):

```
event.preventDefault()  отмена действия браузера в eventListener
onclick="return false"  отмена действия браузера, если событие назначено через 'on'
```

Как использовать
Например, можно как-то обработать ссылки (в примере ниже – вызвать alert и показать адрес), но потом отменить действия по умолчанию (перехода по ссылке не будет):

```
menu.onclick = function(event) {
  if (event.target.nodeName != 'A') return;

  let href = event.target.getAttribute('href');
  alert( href ); // может быть подгрузка с сервера, генерация интерфейса и т.п.

  return false; // отменить действие браузера (переход по ссылке)
};
```

eventListener {passive: true}

Необязательная опция «passive: true» для addEventListener сигнализирует браузеру, что обработчик не собирается выполнять preventDefault().
Опция passive: true сообщает браузеру, что обработчик не собирается отменять прокрутку. Тогда браузер начинает её немедленно, обеспечивая максимально плавный интерфейс, параллельно обрабатывая событие.
Подробнее и как использовать – в учебнике.

event.defaultPrevented

Свойство event.defaultPrevented установлено в true, если действие по умолчанию было предотвращено, и false, если нет.
Можно использовать вместо прерывания всплытия, чтобы просигналить другим обработчикам, что событие обработано.

Например, вместо прерывания всплытия, родительский элемент может получать всплывшее событие и проверять значение .defaultPrevented. Если оно false, то прерывать дальнейшую обработку.
См. пример в учебнике.

Генерация пользовательских событий

Event конструктор

Конструктор создает новый объект события Event.
Документация тут.

```
event = new Event( typeArg, eventInit{} );

typeArg
имя события, строка.

eventInit { (необязательный)
  "bubbles": false,     будет  ли объект всплывающим, true или false
  "cancelable": false,  можно ли отменить действие по умолчанию
  "composed": false     будет ли событие всплывать наружу за пределы shadow root
}
```

Что такое shadow root разбирается в учебнике здесь.

elem.dispatchEvent(event)

Отправляет событие в общую систему событий.
Документация тут.

Если я правильно понял, это именно «срабатывание» события через код.
После того, как объект события создан (через конструктор?), мы должны запустить его на элементе, вызвав метод elem.dispatchEvent(event). Затем обработчики отреагируют на него, как будто это обычное браузерное событие. Если при создании указан флаг `bubbles`, то оно будет всплывать.

Возвращаемое значение — false, если событие отменяемое и хотя бы один из обработчиков этого события вызвал Event.preventDefault().
В ином случае — true.

```
cancelled = !target.dispatchEvent(event)
```

event.isTrusted

Можно легко отличить «настоящее» событие от сгенерированного кодом.

Свойство event.isTrusted принимает значение true для событий, порождаемых реальными действиями пользователя, и false для генерируемых кодом.
Документация тут.

```
var bool = event.isTrusted;
```

MouseEvent, KeyboardEvent и другие

Для некоторых конкретных типов событий есть свои специфические конструкторы. Стоит использовать их вместо new Event, потому что специфический конструктор позволяет указать стандартные свойства для данного типа события.

UIEvent
FocusEvent
MouseEvent
WheelEvent
KeyboardEvent

new CustomEvent

event.preventDefault()

Вложенные события обрабатываются синхронно

</>

Интерфейсные события

Основы событий мыши

Одно действие может вызвать несколько событий.
Клик мышью вначале вызывает mousedown, затем mouseup и click, когда кнопка отпущена.
Обработчики событий вызываются в соответствующем порядке: mousedown - mouseup - click

Простые:

mousedown	кнопка мыши нажата на элементе
mouseup	кнопка мыши отжата на элементе
mouseover	мышь наведена на элемент
mouseout	мышь покидает элемент
mousemove	движение мыши над элементом
contextmenu	открытие контекстного меню (пкм обычно, это не совсем событие мыши)

Комплексные:

click	клик на элементе левой кнопкой или тап на сенсорном экране. mousedown + mouseup
dblclick	двойной клик
oncopy	запрет копирования, если запретить действие браузера по умолчанию.

MouseEvent.which

Документация: MouseEvent.which
Свойство показывает, какая именно кнопка мыши вызвала событие MouseEvent.
Для мыши настроенной для левшей порядок значений будет изменён.
Не стандартизированно. Стандартная альтернатива этому свойству – MouseEvent.button и MouseEvent.buttons.

```
let buttonPressed = MouseEvent.which

0 - нет кнопки
1 - левая
2 - средняя (если есть)
3 - правая
```

Модификаторы: shift, alt, ctrl и meta

Все события мыши включают в себя информацию о нажатых клавишах-модификаторах. У объекта событий есть свойства, которые соответствуют кнопкам-модификаторам. Если при клике кнопка была зажата, её свойство будет равняться true.

```
.shiftKey
.altKey
.ctrlKey
.metaKey

let shiftKeyPressed = MouseEvent.shiftKey
true / false
```

Внимание:
Обычно на Mac используется клавиша Cmd вместо Ctrl.
В Windows и Linux клавишами-модификаторами являются Alt, Shift и Ctrl. На Mac есть ещё Cmd, которой соответствует свойство .metaKey.
если мы хотим поддерживать такие комбинации, как Ctrl+клик, то для Mac имеет смысл использовать Cmd+клик.
Вместе с ctrlKey нужно проверять metaKey:

```
if (event.ctrlKey || event.metaKey)
```

Координаты: clientX/Y, pageX/Y

Все события мыши имеют координаты двух видов:

```
Координаты мыши

clientX   координаты
clientY   относительно окна

pageX     координаты
pageY     относительно документа
```

Отключаем выделение

Побочный эффект двойного клика или зажатой левой кнопки – выделение текста.
Этот эффект можно отключить через отмену действий браузера по умолчанию:
(подробнее о запрете отключения в учебнике в главе Selection и Range)

Важно!
Отменять выделение надо именно через «mousedown», не через двойной клик.

```
<b ondblclick="alert('Клик!')" onmousedown="return false">
  Сделайте двойной клик на мне
</b>
```

Движение мыши

Все свойства MouseEvent.

События mouseover, mouseout

mouseover	курсор оказывается над элементом. Документация тут.
mouseout	курсор уходит с элемента. Документация тут.

У этих двух событий есть свойство .relatedTarget.

ev.relatedTarget

Второстепенная цель события, если таковая есть. Используется только с определённым перечнем событий.

Документация тут.

```
для события mouseover:
me.target            элемент, на который курсор перешёл
me.relatedTarget     элемент, с которого курсор ушёл

для события mouseout:
me.target            с которого курсор ушёл
me.relatedTarget     на который курсор перешёл
```

Свойство relatedTarget может быть null.
Такое может быть, если указатель мыши перешёл из-за пределов окна браузера или ушёл туда.
Если, например, написать event.relatedTarget.tagName, то при отсутствии event.relatedTarget будет ошибка.

Пропуск элементов

Событие mousemove не генерируется на каждом пикселе. Если пользователь двигает мышкой очень быстро, то некоторые DOM-элементы могут быть пропущены.
Браузер периодически проверяет позицию курсора и, заметив изменения, генерирует событие mousemove.

Если был mouseover, то будет и mouseout.
Если указатель «официально» зашёл на элемент, то есть было событие mouseover, то при выходе с него обязательно будет mouseout.

При переходе на потомка

По логике браузера, курсор мыши может быть только над одним элементом в любой момент времени – над самым глубоко вложенным и верхним по z-index.
Если курсор переходит на другой элемент (пусть даже дочерний), то он покидает предыдущий.
Поэтому, если перевести курсор на потомка, то его родителю присваивается mouseout.

Важная деталь:
Событие mouseover, происходящее на потомке, всплывает. Поэтому если на родительском элементе есть такой обработчик, то он его вызовет.

Чтобы этого избежать, можно смотреть на relatedTarget и, если мышь всё ещё внутри элемента, то игнорировать такие события.
Или же можно использовать другие события: mouseenter и mouseleave. С ними такая проблема не возникает.

События mouseenter и mouseleave

mouseenter
mouseleave

Генерируются, когда курсор мыши переходит на элемент или покидает его. При этом:
- переходы внутри элемента, на его потомки и с них, не считаются
- они не всплывают, поэтому их нельзя делегировать

Делегирование событий

События «mouseenter» и «leave» не всплывают, поэтому их нельзя делегировать.

Суть делегирования при использовании «mouseover mouseout»:
- запоминать текущий элемент, над которым совершается действие
- игнорировать переходы на его потомков
- игнорировать уход с его потомков

```
let currentElem = null;


function enter(ev) {
  if (currentElem) return;

  // целевой элемент - только ячейка .child
  // если элемент вложен в .child, ничего не делать
  let target = ev.target.closest('.child');
  if (!target) return;

  // если .child не принадлежит текущему parent
  // if (!parent.contains(target)) return;

  currentElem = target;
  target.style.backgroundColor = 'palevioletred';
}


function exit(ev) {
  if (!currentElem) return;

  let relatedTarget = ev.relatedTarget;

  // поднимаемся по дереву элементов и проверяем – внутри ли мы currentElem или нет
  // если да, то это переход внутри элемента – игнорируем
  while (relatedTarget) {
    if (relatedTarget == currentElem) return;
    relatedTarget = relatedTarget.parentNode;
  }

  // мы действительно покинули элемент
  currentElem.style.background = '';
  currentElem = null;
}


parent.addEventListener('mouseover', enter);
parent.addEventListener('mouseout', exit);
```

```
<div class="parent" id="parent">
  <div class="child">
    <div class="child__inner"></div>
  </div>

  <div class="child">
    <div class="child__inner"></div>
  </div>
</div>
```

```
.parent {
  display: flex;
  justify-content:space-evenly;
  align-items: center;
  width: 400px;
  height: 200px;
  border: 1px solid gray;
}

.child {
  display: flex;
  justify-content: center;
  align-items: center;
  width: 150px;
  height: 150px;
  background-color: paleturquoise;
}

.child__inner {
  width: 50px;
  height: 50px;
  background-color: peachpuff;
}
```

Drag'n'Drop с событиями мыши

Drag and drop в интерфейсах веб API

События Drag (тут):
dragstart
Событие dragstart вызывается, когда пользователь начинает перетаскивать выделенный элемент или текст. Обработчик может быть использован для сохранения информации о перемещаемом объекте, а также для изменения изображения, которое будет ассоциировано с перемещением.

Если я правильно понял, глава посвящена самостоятельной реализации drug'n'drop и не использует API.
Алгоритм:
1-е событие – «mousedown»
Приготовить элемент к перемещению (например, создать его копию)

2-е событие – «mousemove»
Передвинуть элемент на новые координаты через «left top»

3-е событие – «mouseup»
Остановить перенос элемента, произвести действия, связанные с окончанием Drag’n’Drop.

Ещё важные действия:
- Браузер имеет свой Drag’n’Drop, поэтому чтобы они не конфликтовали, его надо отключить.
- Браузер отслеживает mousemove с определённым интервалом, поэтому при быстром движении мышью можно выскочить за пределы нужной формы. Поэтому надо отслеживать «mousemove» на всём document.
Правильнее этот вопрос решать через el.setPointerCapture(pointerId).

Пример:
В документе 2 объекта:
- droppable, на который перетаскиваются объекты;
- draggable, перетаскиваемый объект.
При перемещении «draggable» на «droppable», у «draggable» меняется цвет фона.

```
ball.ondragstart = () => false;  // отключить встроенный в браузер DnD
let currentDroppable = null;     // объект, который предназначается для "сброса"

// 1
ball.onmousedown = function(event) {
  // 2
  let shiftX = event.clientX - ball.getBoundingClientRect().left;
  let shiftY = event.clientY - ball.getBoundingClientRect().top;

  // 3
  ball.style.position = 'absolute';
  ball.style.zIndex = 1000;
  document.body.append(ball);

  // 4
  function moveAt(pageX, pageY) {
    ball.style.left = pageX - shiftX + 'px';
    ball.style.top = pageY - shiftY + 'px';
  }
  // moveAt(event.pageX, event.pageY);


  // 5
  function onMouseMove(event) {
    moveAt(event.pageX, event.pageY);
    // 5.1
    ball.hidden = true;
    let elemBelow = document.elementFromPoint(event.clientX, event.clientY);
    ball.hidden = false;
    if (!elemBelow) return;

    // 5.2
    let droppableBelow = elemBelow.closest('.droppable');

    if (currentDroppable != droppableBelow) {
      if (currentDroppable) {
        // null если мы были не над droppable до этого события
        // (например, над пустым пространством)
        currentDroppable.style.background = '';
      }

      currentDroppable = droppableBelow;
      if (currentDroppable) {
        // null если мы не над droppable сейчас, во время этого события
        // (например, только что покинули droppable)
        currentDroppable.style.background = 'powderblue';
      }
    }
  }

  // 6
  document.addEventListener('mousemove', onMouseMove);

  // 7
  ball.onmouseup = function() {
    document.removeEventListener('mousemove', onMouseMove);
    ball.onmouseup = null;
  };
};
```

1
Самописный DnD вешается на объект, который может перетаскиваться.
Событие «mousedown» создаёт все необходимые функции и запускает их.

2
Переменные «shiftX» и «shifty» нужны для того, чтобы высчитать верхний левый угол (начало координат) у перетаскиваемого объекта и оперировать им: подставлять под координаты курсора.

3
Перетаскиваемый объект становится «absolute» и добавляется в body, чтобы свободно перемещаться везде.

4
Объявление функции «moveAt(pageX, pageY)», которая будет задавать координаты перетаскиваемого объекта при его перемещении. При вызове она прописывает новые значения «left», «right».

5
Объявление ффункции onMouseMove. Она вешается на весь документ с событием «mousemove». Таким образом, она будет много раз запускаться при перемещении объекта.
5.1
Т.к. перетаскиваемый объект перекрывает собой все остальные (находится над ними), он не даёт подсветить площадку для сброса. Чтобы проверить, что находится под ним, его надо на мгновение скрыть, записать в переменную объект под ним, а потом снова показать.

5.2
От объекта под мячом надо подниматься по цепочке вверх и искать предка с классом «droppable».
Если такой…

HTML и CSS для примера:

```
<div class="droppable" id="droppable"></div>
<div class="draggable" id="ball"></div>

<style>
  .droppable {
    width: 150px;
    height: 150px;
    background-color: papayawhip;
  }

  .draggable {
    width: 70px;
    height: 70px;
    background-color: blue;
    border-radius: 50%;
  }
</style>
```

Клавиатура: keydown и keyup
Важно!

На современных устройствах есть другие способы, кроме клавиатуры, «ввести что-то». Например, распознавание речи или Копировать/Вставить с помощью мыши.

События клавиатуры должны использоваться, если мы хотим обрабатывать взаимодействие пользователя именно с клавиатурой (в том числе виртуальной). К примеру, если нам нужно реагировать на стрелочные клавиши Up и Down или горячие клавиши (включая комбинации клавиш).
Единственным исключением является клавиша Fn, которая присутствует на клавиатуре некоторых ноутбуков. События на клавиатуре для неё нет, потому что она обычно работает на уровне более низком, чем даже ОС

Событие input event
Существует специальное событие input, чтобы отслеживать любые изменения в поле <input>. И оно справляется с такой задачей намного лучше. Мы рассмотрим его позже в главе События: change, input, cut, copy, paste.

Коды и символы можно посмотреть здесь, на w3.

keydown и keyup

keydown – событие при нажатии клавиши или автоповторе.
keyup – при отпускании клавижи

Важно!
Все эти события содержат свойства, в которых записаны клавиши-модификаторы: ctrl, shift, alt и meta

```
event.shiftKey
event.altKey
event.ctrlKey
event.metaKey

let shiftKeyPressed = instanceOfMouseEvent.shiftKey
true / false
```

Так, например, выполняется проверка на комбинацию клавиш ctrl + a

```
document.onkeydown = (ev) => {
  if (ev.code == 'KeyA' && (ev.ctrlKey || ev.metaKey) ) {
    alert('!');
  }
}
```

Свойства этих событий:
ev.code
Получить код клавиши. Точно указывает, какая именно клавиша нажата: ControlLeft, ControlRight.
Важно! Возвращаемое значение всегда начинается с верхнего регистра.

```
document.onkeydown = (ev) => console.log(ev.code);

буквенные клавиши начинаются с Key
KeyZ

цифровые начинаются с Digit
Digit7

код специальных – их наименование
ControlLeft, ControlRight
F6
```

ev.key
Получить символ. Мб быть в верхнем и нижнем регистрах.

ev.repeat
Автоповтор. Происходит, если зажать клавишу и не отпускать. Для событий, вызванных автоповтором, у объекта события свойство event.repeat равно true.

Действия по умолчанию

- появление символа
- удаление символа (клавиша Delete).
- прокрутка страницы (клавиша PageDown).
- открытие диалогового окна браузера «Сохранить» (Ctrl+S)
- другое

event.preventDefault() работает практически всегда. Не работает на уровне операционной системы. Например, Alt+F4 закроет браузер в Windows и это не отменить.

Пример:
Фильтр для телефонного поля.

```
<input id="input" placeholder="Введите телефон" type="tel">


function checkKey(ev) {
  let correctKeys = [ '+', 'Delete', 'Backspace', 'ArrowLeft', 'ArrowRight'];
  // выполняется проверка на допустимый символ
  let result = (0 <= ev.key && ev.key <= 9) || correctKeys.includes(ev.key);
  // если проверка не прошла, то отменить действие по умолчанию (печать)
  if (!result) ev.preventDefault();
}

input.addEventListener('keydown', checkKey);
```

В учебнике была другая реализация. Код прописан прямо в HTML и проверка возвращает true или false. Если вернётся false, то это отменяет действие по умолчанию.

Тем не менее, такой фильтр не надёжен, потому что можно вставить что-то мышью.
Альтернатива – отслеживать событие input, оно генерируется после любых изменений в поле <input>, и мы можем проверять новое значение и подчёркивать/изменять его, если оно не подходит.

Старые свойства

В прошлом существовало событие **keypress** и его свойства: keyCode, charCode, which. Сейчас они объявлены устаревшими. Старый код ещё работает, так как браузеры продолжают поддерживать и keypress, и keyCode с charCode, и which, но более нет никакой необходимости в их использовании.

События указателя

События указателя (Pointer events) – это современный единый способ обработки ввода с помощью различных указывающих устройств: мышь, стилус, сенсорный экран и других.

События указателя

```
API PointerEvent

pointerdown
pointerup
pointermove
pointerover
pointerout
pointerenter
```

```
pointerleave

pointercancel
gotpointercapture
lostpointercapture
```

До «pointercancel» есть аналоги у mouseEvent.
Можно заменить события «mouse» на аналогичные «pointer», с мышью по-прежнему всё будет работать нормально, при этом поддержка сенсорных устройств улучшится. Кое-где понадобится добавить #? **touch-action**: none в CSS.

Свойства событий указателя

События указателя содержат те же свойства, что и события мыши (потому что наследуют), например clientX/Y, target и т.п., и несколько дополнительных.

Все свойства MouseEvent + дополнительные свойства здесь:

```
pe.pointerId    уникальный идентификатор указателя, вызвавшего событие (м.б. несколько)
pe.pointerType  тип указывающего устройства: «mouse», «pen» или «touch»
pe.isPrimary    true для основного указателя (первый палец в мульти-тач)

pe.width        ширина области соприкосновения указателя (пальца). Если не поддерживается: 1
pe.height       высота области соприкосновения указателя. Если не поддерживается: 1
pe.pressure     степень давления указателя от 0 до 1. Если не поддерживают: 0.5 либо 0

pe.tiltX        специфичные для пера свойства
pe.tiltY
pe.twist

t.tangentialPressure нормализованное тангенциальное давление
```

Мульти-тач

Мультитач – это возможность касаться сразу нескольких мест на телефоне или планшете или выполнять специальные жесты. В этом случае используются свойства событий «pointerId» и «isPrimary».

При касании **первым пальцем**:
происходит событие pointerdown со свойством isPrimary=true и некоторым pointerId.
События, связанные с первым пальцем, всегда содержат свойство isPrimary=true.

При касании **вторым и последующими** пальцами:
происходит событие pointerdown со свойством isPrimary=false и уникальным pointerId для каждого касания.

Мы можем отслеживать несколько касающихся экрана пальцев, используя их pointerId. Когда пользователь перемещает, а затем убирает палец, получаем события pointermove и pointerup с тем же pointerId, что и при событии pointerdown.

Событие pointercancel

Событие происходит, когда текущее действие с указателем по какой-то причине прерывается, и события указателя больше не генерируются, например:
- Указывающее устройство было физически выключено.
- Изменилась ориентация устройства (перевернули планшет).
- Браузер решил сам обработать действие, считая его жестом мыши, масштабированием и т.п.

Например, когда реализуешь перетаскиваие самостоятельно, а браузер его отменяет и запускает встроенный DnD. **Предотвращайте действие браузера по умолчанию**, чтобы избежать pointercancel**.**

Нужно сделать две вещи:
1. Предотвратить запуск встроенного drag’n’drop
- задать ball.ondragstart = () => false.
2. Для сенсорного экрана:
- добавить в CSS свойство #ball { touch-action: none }.

el.setPointerCapture(pointerId)

Документация тут.
Привязывает события с данным pointerId к elem. После этого все события указателя с таким pointerId будут иметь elem в качестве целевого элемента (как будто произошли над elem), вне зависимости от того, где в документе они произошли. В результате элемент продолжит получать события указателя, даже если указатель вышел за его пределы. Другими словами, меняет target всех дальнейших событий с данным pointerId на elem.

Эта привязка отменяется:
- автоматом, при возникновении события pointerup или pointercancel,
- автоматом, если elem удаляется из документа,
- при вызове elem.releasePointerCapture(pointerId).

```
targetElement.setPointerCapture(pointerId);
```

Существует два связанных с захватом события:
gotpointercapture
срабатывает, когда элемент использует setPointerCapture для включения захвата.

lostpointercapture
срабатывает при освобождении от захвата: явно с помощью releasePointerCapture или автоматически, когда происходит событие pointerup/pointercancel.

Element.releasePointerCapture()

.PointerCapture, пример

Это особая возможность событий указателя.

В одной из задач реализовывался бегунок. Чтобы мышь не перескакивала за его границы во время перетаскивания, событие назначалось на весь Document. См. здесь.
Одна из проблем – это то, что движения указателя по документу могут вызвать сторонние эффекты, заставить сработать другие обработчики, не имеющие отношения к слайдеру.

С помощью .setPointerCapture() это можно сделать проще.
Захват указателя позволяет привязать pointermove к thumb и избежать любых подобных проблем:
1. можно вызывать thumb.setPointerCapture(event.pointerId) в обработчике pointerdown.
2. дальнейшие события указателя (до pointerup/cancel) будут привязаны к thumb.
3. когда произойдёт pointerup (передвижение завершено), привязка будет автоматически удалена.

Таким образом, мы имеем два бонуса:
1. Код становится чище, поскольку нам больше не нужно добавлять/удалять обработчики для всего документа. Удаление привязки происходит автоматически.
2. Если в документе есть какие-то другие обработчики pointermove, то они не будут нечаянно вызваны, пока пользователь находится в процессе перетаскивания слайдера.

```
thumb.onpointerdown = function(event) {
    // все события указателя перенаправить на thumb (пока не произойдёт 'pointerup')
    thumb.setPointerCapture(event.pointerId);
```

```
  };

  thumb.onpointermove = function(event) {
    // перемещение ползунка: все события перенаправлены на этот обработчик
    let newLeft = event.clientX - slider.getBoundingClientRect().left;
    thumb.style.left = newLeft + 'px';
  };

  // не надо вызывать thumb.releasePointerCapture,
  // при срабатывании события 'pointerup' это происходит автоматически
```

Прокрутка

```
Прокрутка
window.pageYOffset    показать кол-во пикселей, на которое прокручено окно по Y. Только чтение
window.pageXOffset    показать кол-во пикселей, на которое прокручено окно по X. Только чтение

window.scroll(x, y)   прокрутить страницу до указанного места
window.scrollTo(x, y) тот же самый эффект
window.scrollBy(x, y) прокручивает документ на указанные величины

window.scrollByPages() Прокручивает документ на указанное число страниц
window.scrollByLines() Прокручивает документ на заданное число строк


Событие
scroll                Событие возникает при прокрутке области просмотра документа или элемента.
onscroll
```

Событие прокрутки scroll позволяет реагировать на прокрутку страницы или элемента.
Событие scroll работает как на window, так и на других элементах, на которых включена прокрутка.
Так, например, можно показывать и скрывать хеддер, или подгружать данные по мере прокрутки страницы.

scroll, .onscroll

Событие scroll возникает при прокрутке области просмотра документа или элемента.

```
element.addEventListener('scroll', foo)
element.onscroll = functionReference
```

Поскольку события прокрутки могут запускаться с высокой скоростью, обработчик событий не должен выполнять вычислительно-ёмкие операции, такие как модификации DOM. Вместо этого рекомендуется пропускать такты события, используя setTimeout().

функция для отображения текущей прокрутки:

```
window.addEventListener('scroll', function() {
  document.getElementById('showScroll').innerHTML = pageYOffset + 'px';
});


document.addEventListener('scroll', () => {
  div.innerText = pageYOffset;
});
```

Window.pageYOffset

Документация тут.
Свойство окна Window только для чтения.
Возвращает количество пикселей, на которое прокручен документ по вертикали (вниз или вверх). Значение - число с плавающей запятой. Значение равное 0.0 означает, что вертикальная прокрутка ещё не была совершена.
Т.к. это свойство глобального объекта window, можно просто указывать pageYOffset.

Свойство соответствует Window.scrollY. Они идентичны, но есть незначительная разница в поддержке pageYOffset и scrollY: первый поддерживается лучше в старых браузерах, использовать можно любое свойство.

```
yOffset = window.pageYOffset;
```

window.pageXOffset

Документация тут.
Возвращает количество пикселей, на которое документ прокручен по горизонтали, это то же самое, что и scrollX.

```
xOffset = window.pageXOffset;
```

window.scroll(x, y)

Документация тут.
Прокручивает страницу до указанного места.

```
window.scroll(x-coord,y-coord)
```

window.scrollTo(x, y)

Документация тут.
То же самое, что и w.scroll

window.scrollBy()

Прокручивает документ на указанные величины.

```
window.scrollBy(X, Y);

// Прокрутка на один экран вертикально вниз
window.scrollBy(0, window.innerHeight);
```

window.scrollByPages()

Прокручивает документ на указанное число страниц. Не стандартизированно.

window.scrollByLines()

Прокручивает документ на заданное число строк. Не стандартизированно.

Предотвращение прокрутки

Нельзя предотвратить прокрутку, используя event.preventDefault() в обработчике onscroll, потому что он срабатывает *после* того, как прокрутка уже произошла.
Но можно предотвратить прокрутку, используя event.preventDefault() на событии, которое вызывает прокрутку, например, на событии keydown для клавиш pageUp и pageDown.

Если поставить на них обработчики, в которых вызвать event.preventDefault(), то прокрутка не начнётся.
Способов инициировать прокрутку много, поэтому более надёжный способ – использовать CSS, свойство **overflow**.

Примеры

Бесконечная прокрутка

```
function populate() {
  while(true) {
    let windowRelativeBottom = document.documentElement.getBoundingClientRect().bottom;
    if (windowRelativeBottom > document.documentElement.clientHeight + 100) break;
    document.body.insertAdjacentHTML("beforeend", `<p>Date: ${new Date()}</p>`);
  }
}

window.addEventListener('scroll', populate);

populate(); // инициализация документа
```

windowRelativeBottom
Это координата нижней границы документа относительно окна. Пожалуйста, обратите внимание, что bottom не может быть 0, потому что низ документа никогда не достигнет верха окна. Нижним пределом координаты bottom является высота самого окна, больше прокручивать вверх нельзя.

По условиям задачи, windowRelativeBottom = «высота окна» + 100px.

document.documentElement.clientHeight
Это высота окна браузера.

Подгрузка изображений
Загрузка видимых изображений
</>

Формы, элементы управления

Свойства и методы формы

Формы и её элементы, такие как <input>, имеют множество специальных свойств и событий.

Коллекция document.forms

document.forms
Возвращает коллекцию (HTMLCollection) форм в текущем документе.
Если на странице форм нет, тогда возвращённый результат будет пустым, а длина коллекции равна нулю.

Формы в документе входят в специальную коллекцию document.forms. Это так называемая «именованная» коллекция. В коллекции они доступны через:
- имя формы
- порядковый номер:

```
document.forms.my   форма с именем "my" (name="my")
document.forms[0]   первая форма в документе
```

Свойство form.elements

HTMLFormElement.elements
возвращает коллекцию элементов внутри формы (вне зависимости от вложенности).

Элементы внутри формы доступны по именам через свойство .elements.
Имя – это атрибут name=" "
Все элементы, вне зависимости от глубины вложенности, доступны сразу в .elements. Т.е. не надо прописывать всю цепочку вложенности, чтобы достать элемент.
Если в форме несколько элементов с одинаковым именем, то form.elements[name] будет коллекцией.
Элементы <fieldset> также содержат свойство elements.

```
document.forms[0].elements.name
```

Сокращённая форма записи
вместо «form.elements.login» можно написать «form.login».
Есть небольшая проблема: если получить элемент, а затем изменить его свойство name, то он будет доступен и под старым, и под новым именем.

Свойство element.form

HTMLSelectElement.form
В свою очередь, все элементы формы ссылаются на неё через своё свойство .form

```
element.form === form
```

input

Документация по элементу тут.
HTMLInputElement
input.**value**
Строка. Значение input.

```
input.value = "Новое значение";
textarea.value = "Новый текст";
```

input.**checked**
Булево значение. Когда input – это radio или checkbox, означает наличие или отсутствие выбора.

```
input.checked = true; // для чекбоксов и переключателей
```

textarea

Документаци по элементу тут.
textarea.**value**
Строка. Надо использовать .value вместо .innerHTML.

```
input.value = "Новое значение";
textarea.value = "Новый текст";
```

select

Документация тут.

```
select.options         коллекция из подэлементов <option>,
```

```
select.value           значение выбранного в данный момент <option>,
select.selectedIndex   индекс выбранного <option>.
```

Они дают три разных способа установить значение в <select>:
1. Найти соответствующий элемент <option> и установить в option.selected значение true.
2. Установить в select.value значение нужного <option>.
3. Установить в select.selectedIndex номер нужного <option>.

```
<select id="select">
  <option value="apple"> Яблоко </option>
  <option value="pear"> Груша </option>
  <option value="banana"> Банан </option>
</select>

<script>
  // все три строки делают одно и то же
  select.options[2].selected = true;
  select.selectedIndex = 2;
  select.value = 'banana';
</script>
```

<select> позволяет выбрать несколько вариантов одновременно, если у него стоит атрибут multiple. В этом случае для работы со значениями необходимо ставить или удалять свойство selected у подэлементов <option>.
Их коллекцию можно получить как select.options

option

Документация тут.

option.**selected**
Выбрана ли опция.

option.**index**
Номер опции среди других в списке <select>.

option.**text**
Содержимое опции (то, что видит посетитель).

new Option

```
option = new Option(text, value, defaultSelected, selected);

text              текст внутри <option>,
value             значение
defaultSelected   если true, то ставится HTML-атрибут selected,
selected          если true, то элемент <option> будет выбранным.
```

defaultSelected задаёт HTML-атрибут, его можно получить как option.getAttribute('selected'),
а selected – выбрано значение или нет, именно его важно поставить правильно.
Обычно ставят оба этих значения в true или не ставят вовсе (т.е. false).

Фокусировка: focus/blur

Фокус – это когда пользователь кликает по элементу или использует Tab и находится «в элементе».

Blur – это потеря фокуса, когда пользователь куда-то ещё кликает или жмёт Tab дальше.

Потеря фокуса обычно означает «данные введены» и теперь можно выполнить проверку введённых данных или отправить эти данные на сервер.

Причин потери фокуса много.
Например, когда посетитель кликает куда-то ещё. JavaScript тоже может быть причиной. Из-за этих особенностей обработчики focus/blur могут сработать тогда, когда это не требуется. Используя эти события, нужно быть осторожным.
- alert переводит фокус на себя – элемент теряет фокус (происходит событие blur), а когда alert закрывается – элемент получает фокус обратно (событие focus).
- если элемент удалить из DOM, фокус также будет потерян. Если элемент добавить обратно, то фокус не вернётся.

События focus, blur

```
focus    событие вызывается в момент фокусировки
blur     событие вызывается в момент потери фокуса
```

У focus и blur есть аналоги: focusin и focusout соответсвенно. Между ними разница в том, что аналоги всплывают, а фокус и блюр нет, но доступны только вниз на фазе перехвата.
Эти события должны использоваться с elem.addEventListener, но не с on<event>.

Пример.
Современный HTML позволяет делать валидацию с помощью атрибутов «required», «pattern».
JavaScript можно использовать, когда мы хотим больше гибкости.
Есть поле ввода эл. почты.
Blur: проверяет поле ввода и если пользователь ввёл недопустимый символ – показать ошибку,
Focus: если пользователь зашёл в поле ввода адреса, удалить ошибку.

```
<style>
  .invalid { border-color: red; }
  #error { color: red }
</style>

Ваш email: <input type="email" id="input">
<div id="error"></div>


JS код
input.onblur = () => {
  if (!input.value.includes('@')) {
    input.classList.add('invalid');
    error.innerHTML = 'Введите правильный email';
  }
};

input.onfocus = function() {
  if (this.classList.contains('invalid')) {
    this.classList.remove('invalid');
    error.innerHTML = '';
  }
};
```

Методы focus, blur

```
HTMLElement.focus()  установить фокус на элементе
HTMLElement.blur()   снять фокус с элемента
```

Так, например, можно запретить посетителю переключаться с поля ввода, если введённое значение не прошло валидацию (не работает в Firefox):

```
<script>
  input.onblur = function() {

    if (!this.value.includes('@')) { // не email
      // показать ошибку
      this.classList.add("error");
      // ...и вернуть фокус обратно
      input.focus();

    } else {
      this.classList.remove("error");
    }
  };
</script>
```

tabindex, фокусировка на любом элементе

Поддерживают focus/blur элементы, с которыми пользователь взаимодействует: <button>, <input>, <select>, <a> и т.д. Не поддерживают focus/blur элементы форматирования: , , <table>.

HTML-атрибут tabindex может что угодно сделать фокусируемым. Его значение – это порядковый номер элемента, когда Tab используется для переключения между элементами.

tabindex
Глобальный атрибут. Целое число, определяющее порядок последовательной навигации (при нажатии Tab).

Можно прописывать и как атрибут, и как dom-свойство: el.tabIndex

Положительное – порядок фокусировки определён этим значением. В случае, если несколько элементов содержат одно и то же значение tabindex, их порядок навигации определён расположением в документе (DOM).

0 - порядок навигации определён платформой (как если бы индекса не было).
Отрицательное – может быть выделен, но не участвует в последовательной навигации. Клавиша Tab проигнорирует такой элемент, но метод elem.focus() будет действовать.

```
<li tabindex="1">Один</li>

elem.tabIndex = 1
```

Document.activeElement

Возвращает текущий сфокусированный элемент, то есть элемент, на котором будут вызываться события клавиатуры, если пользователь начнёт с неё ввод. Этот атрибут доступен только для чтения.

```
var curElement = document.activeElement;
```

el.autofocus

HTMLSelectElement.autofocus
HTML-атрибут autofocus устанавливает фокус на элемент, когда страница загружается.

```
aBool = aSelectElement.autofocus;  // Get the value of autofocus
aSelectElement.autofocus = aBool;  // Set the value of autofocus
```

События: change, input, cut, copy, paste

Change и input – это события, сопутствующие обновлению данных.
Интерфейс ClipboardEvent представляет события, связанные с изменением буфера обмена: cut, copy и paste.

change

HTMLElement: change event
Событие срабатывает по окончании изменения элемента.
Для текстовых <input> он сработает при потере фокуса.
Для select, input type=checkbox, radio он сработает сразу после изменения значения.

```
<input type="text" onchange="alert(this.value)">

<select onchange="alert(this.value)">
  <option value="">Выберите что-нибудь</option>
  <option value="1">Вариант 1</option>
  <option value="2">Вариант 2</option>
  <option value="3">Вариант 3</option>
</select>
```

input

HTMLElement: input event
Срабатывает каждый раз при изменении значения.
В отличие от событий клавиатуры, оно работает при любых изменениях значений: вставка с помощью мыши или распознавание речи при диктовке текста.

Речь идёт именно об изменении значения, событие не происходит, если не меняется значение в текстовом поле, т.е. нажатия клавиш ⇦, ⇨ и подобных при фокусе на текстовом поле не вызовут это событие.

Так как input происходит после изменения значения, нельзя ничего предотвратить в oninput.
Использовать event.preventDefault() бесполезно, будет уже слишком поздно.

```
<input type="text" id="input"> oninput: <span id="result"></span>

<script>
  input.oninput = function() {
    result.innerHTML = input.value;
  };
</script>
```

Cut

Copy

Paste

События происходят при вырезании, копировании, вставке данных.
Относятся к классу ClipboardEvent (о нём ниже) и обеспечивают доступ к копируемым и вставляемым данным.

Существует список методов в спецификации для работы с различными типами данных, чтения/записи в буфер обмена.

Можно использовать event.preventDefault() для предотвращения действия по умолчанию, в итоге ничего не скопируется и не вставится. Пример ниже предотвращает все подобные события и показывает, что мы пытаемся сделать:

```
<input type="text" id="input">

<script>
  input.oncut = input.oncopy = input.onpaste = function(event) {
    alert(event.type + ' - ' + event.clipboardData.getData('text/plain'));
    return false;
  };
</script>
```

.clipboardData

ClipboardEvent – это API представляет события, которые предоставляют информацию об изменении буфера обмена: cut, copy и paste.

ClipboardEvent.clipboardData
Является DataTransfer объектом, который содержит данные, полученные от совершения пользователем операции cut, copy или paste.

```
data = ClipboardEvent.clipboardData
```

Что такое **Data transfer**, с которым работает .clipboardData:
Очень мутная тема, вот ссылки:
DataTransfer – это API

.getData()
Возвращает данные для указанного типа или пустую строку, если данные для указанного типа не существуют или передаваемая сущность не содержит данных.

```
dataTransfer.getData(format);

format
тип данных для извлечения: text/plain или text/uri-list
```

Снова о .**clipboardData**
Это свойство (.clipboardData) используется:
- с обработчиками cut и copy, чтобы указать, какие данные следует поместить в буфер обмена с помощью вызова setData(format, data);
- с обработчиком paste, чтобы указать, какие даные будут вставлены (обычно через getData(format), о котором было написано выше).

Отправка формы: событие и метод submit

MDN:
<form>
Руководство по HTML-формам
Все элементы форм тут

submit событие

При отправке формы срабатывает событие submit event. Оно используется для предварительной проверки формы, после чего отправка формы на сервер либо осуществляется, либо предотвращается.
«submit event» происходит **в двух случаях**:
1. нажать кнопку <input type="submit"> или <input type="image">
2. нажать Enter, находясь на каком-нибудь поле. Если нажать «enter» в текстовом поле, то сгенерируется событие «click» на кнопке <input type="submit">, хотя на самом деле клика не было.

Обработчик может проверить данные, и если есть ошибки, показать их и вызвать event.preventDefault(), тогда форма не будет отправлена на сервер.

submit() метод

.submit()
Метод отправляет форму на сервер вручную, в обход «submit event». При этом само событие «submit event» **не генерируется**. Предполагается, что если вызван метод, то все проверки уже выполнены.

```
HTMLFormElement.submit()
document.forms["myform"].submit();
```

Метод похож на кнопку <button>. У «button» есть атрибут «type = submit», который отправляет данные на сервер (установлен по умолчанию по-моему), но между ними есть различия.
При использовании .submit:
- событие submit не инициируется, а кнопка его инициализирует;
- проверка ограничений, соответственно, не запускается.

Для того, при использовании метода .submit() запускалось «submit event» и выполнялась проверка, надо использовать дополнительно другой метод: .requestSubmit().

</>

Загрузка документа и ресурсов

Страница: DOMContentLoaded, load, beforeunload, unload

Жизненный цикл страницы состоит из следующих событий:

```
DOMContentLoaded
HTML: загружен
DOM-дерево: построено
Внешние ресурсы (img, стили): не загружены
обработчик может искать DOM-узлы и инициализировать интерфейс

load event
+ Внешние ресурсы (img, стили)
внешние ресурсы были загружены, стили применены, размеры картинок известны и т.д.

beforeunload
пользователь покидает страницу
Можно проверить, сохранил ли пользователь изменения и спросить, на самом ли деле он хочет уйти.

unload
пользователь почти ушёл, но всё ещё монжно запустить некоторые операции, например, отправить статистику.
```

load и DOMContentLoaded - это события. Оба срабатывают при загрузке страницы в браузере. Разница в том, что DOMContentLoaded отрабатывает, когда браузер полностью загрузил HTML и было построено DOM-дерево, но внешние ресурсы (стили, скрипты, картинки) ещё не прогружены.
Load – это событие, когда браузер загрузил HTML и все зависимые внешние ресурсы. Из этого следует, что он всегда срабатывает после.

DOMContentLoaded

Document: DOMContentLoaded event

Ставится через addEventListener.

Особенности с тегом **script**
Если встречается тег script, то событие ждёт, пока код выполнится полностью. Это нужно на случай, если скрипт что-то дописывает в документ. Два исключения:
1. скрипты с атрибутом async не блокируют событие;
2. сгенерированные динамически скрипты «document.createElement('script')» не блокируют событие.

Особенности со **внешними стилями**
Событие не ждёт пока подгрузятся внешние стили.
Исключение: если после стилей есть тег script. Потому что script может использовать информацию из стилей: размеры и координаты и другие свойства, которыми может оперировать код.

После наступления события DOMContentLoaded происходит встроенное в браузер автозаполнение полей.

window.load

Событие «load» на объекте «window» наступает, когда загрузилась вся страница, включая стили, картинки и другие ресурсы.
Теперь можно получить размеры картинок и всего остального, что не загрузилось в DOMContentLoaded.

window.unload

Когда посетитель покидает страницу, генерируется событие unload. В этот момент стоит совершать простые действия, не требующие много времени, вроде закрытия связанных всплывающих окон.

It is fired after:
beforeunload (cancelable event)
pagehide

The document is in the following state:
- All the resources still exist (img, iframe etc.)
- Nothing is visible anymore to the end user
- UI interactions are ineffective (window.open, alert, confirm, etc.)
- An error won't stop the unloading workflow

Developers should avoid using this event.
Especially on mobile, the unload event is not reliably fired. For example, the unload event is not fired at all in the following scenario:
1. A mobile user visits your page.
2. The user then switches to a different app.
3. Later, the user closes the browser from the app manager.

Обычно здесь отсылают статистику (клики, прокрутка, просмотры областей страницы). Существует специальный метод «navigator.sendBeacon», описанный в спецификации w3c.
Он посылает данные в фоне. Переход к другой странице не задерживается: браузер покидает страницу, но всё равно выполняет sendBeacon.

Navigator.sendBeacon()
Используется для асинхронной передачи небольшого количества информации поверх HTTP веб-серверу.
Возвращает true, если браузер успешно поставил данные data в очередь отправления, в ином случае false.

При использовании метода sendBeacon(), данные будут переданы на сервер асинхронно, как только браузер найдёт оптимальный момент для этого. Это не вызовет задержек выгрузки и не повлияет на время загрузки следующей страницы. Размер данных ограничен 64 Кб.

```
navigator.sendBeacon(url [, data]);

data
может содержать объект типа ArrayBufferView, Blob, DOMString, или FormData, который будет
передан.
```

Его можно использовать вот так:

```
let analyticsData = { /* объект с собранными данными */ };

window.addEventListener("unload", function() {
  navigator.sendBeacon("/analytics", JSON.stringify(analyticsData));
});
```

beforeunload

Если посетитель собирается уйти со страницы или закрыть окно, обработчик в событии «beforeunload» попросит дополнительное подтверждение.

```
window.onbeforeunload = function() {
  return false;
};
```

У меня сообщение появляется только при перезагрузке страницы, но не при её закрытии.
Если вместо false вернуть не пустую строку, то это будет приравнено к false. Так сделали, чтобы не было злоупотреблений.

document.readyState

Document.readyState
Свойство описывает состояние загрузки document. Состояния: loading interactive complete. Может быть альтернативой событию load.

loading	документ загружается.
interactive	документ был полностью прочитан.
complete	документ был полностью прочитан и все ресурсы (такие как изображения) были тоже загружены.

Так что мы можем проверить document.readyState и, либо установить обработчик, либо, если документ готов, выполнить код сразу же.
Например, вот так:

```
function work() { ... }

if (document.readyState == 'loading') {
  // ещё загружается, ждём события
  document.addEventListener('DOMContentLoaded', work);
} else {
  // DOM готов!
  work();
}
```

Скрипты: async, defer

У скриптов есть несколько особенностей:
1. когда браузер встречает тег «script», он его загружает. Если вверху страницы объёмный скрипт, он блокирует страницу. Пользователи не видят содержимое страницы, пока он не загрузится и не запустится.
2. скрипты не видят DOM-элементы ниже себя, поэтому к ним нельзя добавить обработчики и т.д.

Есть два атрибута тега «script», чтобы решить эту проблему: defer и async.
На практике defer используется для скриптов, которым требуется доступ ко всему DOM и/или важен их относительный порядок выполнения.

А async хорош для независимых скриптов, например счётчиков и рекламы, относительный порядок выполнения которых не играет роли.

HTMLScriptElement

HTMLScriptElement.**defer**
- загружается в фоновом режиме, браузер продолжит обрабатывать страницу.
- выполнится только после готовности DOM-дерева, но до события DOMContentLoaded.
- несколько скриптов загружаются параллельно, но выполняются последовательно, как расположены в документе.
- работает только со внешними скриптами.

HTMLScriptElement.**async**
- загружается в фоновом режиме
- выполнится сразу же, не ждёт DOMContentLoaded.
- несколько скриптов загружаются параллельно и выполняются в порядке загрузки (не ждут друг друга).

Динамически загружаемые скрипты

Мы можем также добавить скрипт и динамически, с помощью JavaScript:

```
let script = document.createElement('script');
script.src = "/article/script-async-defer/long.js";
script.async = false;
document.body.append(script);
```

Скрипт начнёт загружаться, как только он будет добавлен в документ.
Динамически загружаемые скрипты по умолчанию ведут себя как «async». Чтобы это предотвратить, им надо прописать script.async = false.

Загрузка ресурсов: load и error

Для отслеживания загрузки сторонних ресурсов есть два события:
load – успешная загрузка,
error – во время загрузки произошла ошибка.

Вообще, Window: load event – это событие окна, но также может использоваться и с элементами: script.onload.
События load и error также срабатывают и для других ресурсов, а вообще, для любых ресурсов, у которых есть внешний src.

Обработчики onload/onerror отслеживают только сам процесс загрузки. Ошибки обработки и выполнения загруженного скрипта ими не отслеживаются. Чтобы «поймать» ошибки в скрипте, нужно воспользоваться глобальным обработчиком window.onerror.

Загрузка скриптов

```
let script = document.createElement('script');

script.src = "https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.3.0/lodash.js"
document.head.append(script);

script.onload = function() {
  // в скрипте создаётся вспомогательная функция с именем "_"
  alert(_); // функция доступна
};
```

error
Событие возникает, когда произошла какая-либо ошибка. Точные обстоятельства могут быть различными, потому что события с этим именем используются множеством различных API.

```
let script = document.createElement('script');
script.src = "https://example.com/404.js"; // такого файла не существует
document.head.append(script);

script.onerror = function() {
  alert("Error loading " + this.src); // Ошибка загрузки https://example.com/404.js
};
```

</>

Чистые таблицы