**Frontend Questions**

**JS in Browser**

[**Что такое DOM?**](https://youtu.be/1eIRTdgzHtw?t=471)

DOM или Document Object Model (объектная модель документа) — это прикладной программный интерфейс (API) для работы с HTML и XML документами. Когда браузер первый раз читает («парсит») HTML документ, он формирует большой объект, действительно большой объект, основанный на документе — DOM. DOM представляет собой древовидную структуру (дерево документа). DOM используется для взаимодействия и изменения самой структуры DOM или его отдельных элементов и узлов.  
  
Допустим, у нас есть такой HTML:

<!DOCTYPE **html**>

<**html** lang="en">

<**head**>

<**meta** charset="UTF-8">

<**meta** name="viewport" content="width=device-width">

<**meta** http-equiv="X-UA-Compatible" content="ie=edge">

<**title**>Document Object Model</**title**>

</**head**>

<**body**>

<**div**>

<**p**>

<**span**></**span**>

</**p**>

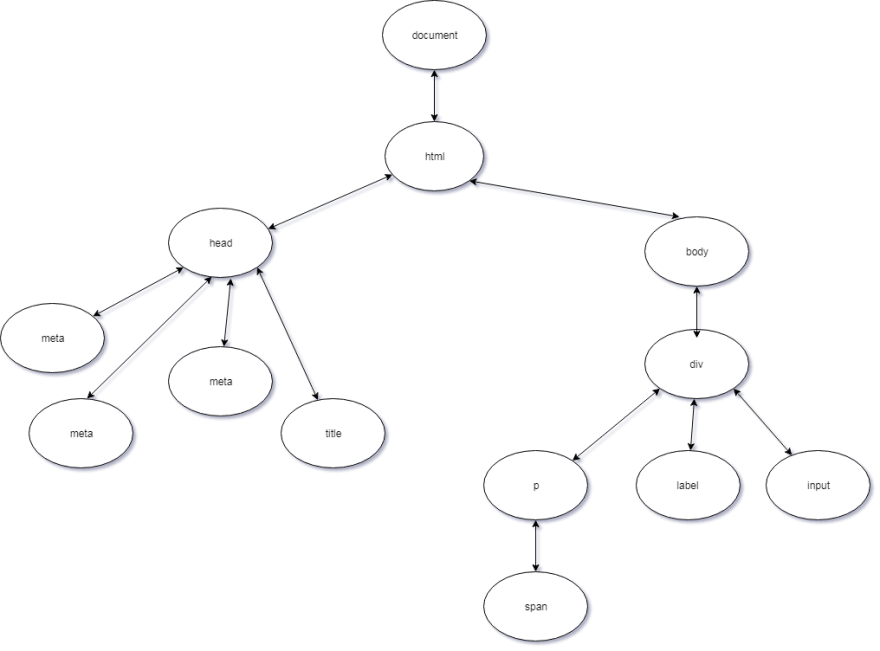
<**label**></**label**>

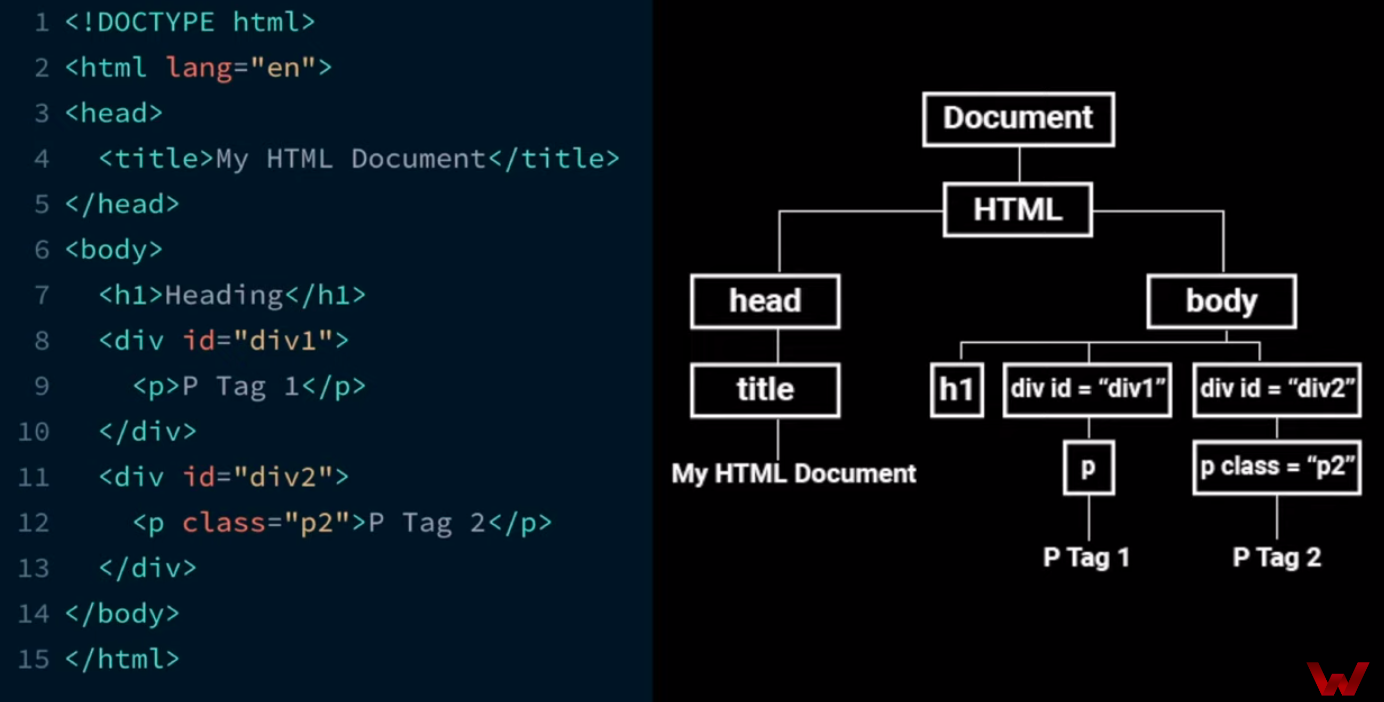
<**input**>

</**div**>

</**body**>

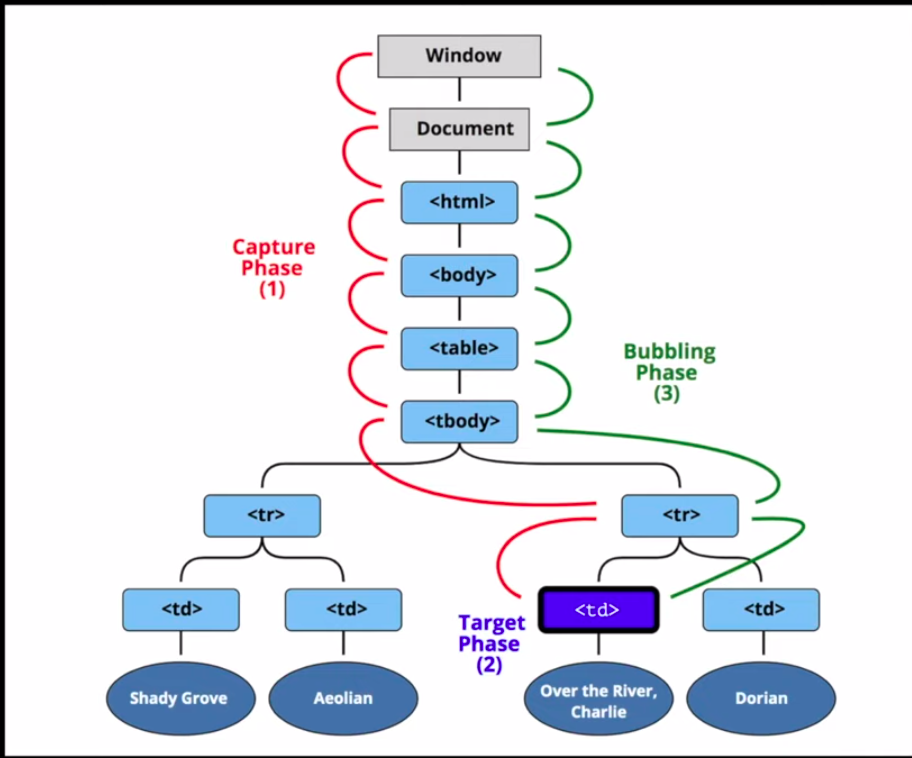
</**html**>

DOM этого HTML выглядит так:  
  
  
  
В JS DOM представлен объектом Document. Объект Document имеет большое количество методов для работы с элементами, их созданием, модификацией, удалением и т.д.



Объектная модель документа, которую браузер создает в памяти компьютера на основании html-кода, полученного им от сервера. Если говорить просто, то html-код – это текст html страницы, а dom – это набор связанных объектов, созданных браузером при парсинге этого текста. По сути, это интерфейс, при помощи которого js может взаимодействовать с элементами страницы. DOM имеет древовидную структуру, по его элементам можно перемещаться, изменять, добавлять события и различные обработчики.

[**Что такое распространение события (Event Propagation)?**](https://youtu.be/1eIRTdgzHtw?t=522)



Это механизм, который отрабатывает, когда какое-либо событие происходит в документе. Событие распространяется от объекта window до вызывающего его элемента. При этом событие последовательно затрагивает всех предков целевого документа. Есть три основные фазы:

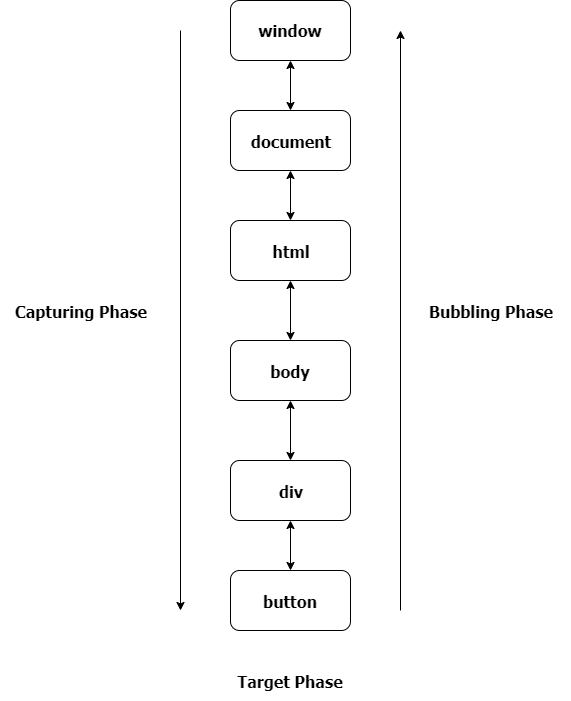
1) capturing – фаза захвата: Событие начинается от корня документа и проходит по dom дереву до целевого элемента.

2) target – событие достигает целевого элемента. Обычно его называют event target.

3) bubbling – фаза всплытия. Событие возвращается обратно до window, параллельно вызывая все события на родительских элементах.

Когда какое-либо событие происходит в элементе DOM, оно на самом деле происходит не только в нем. Событие «распространяется» от объекта Window до вызвавшего его элемента (event.target). При этом событие последовательно пронизывает (затрагивает) всех предков целевого элемента. Распространение события имеет три стадии или фазы:

1. Фаза погружения (захвата, перехвата) — событие возникает в объекте Window и опускается до цели события через всех ее предков.
2. Целевая фаза — это когда событие достигает целевого элемента.
3. Фаза всплытия — событие поднимается от event.target, последовательно проходит через всех его предков и достигает объекта Window.

  
  
Подробнее о распространении событий можно почитать [здесь](https://learn.javascript.ru/bubbling-and-capturing) и [здесь](https://www.w3.org/TR/DOM-Level-3-Events/" \l "event-flow).

#### Что такое всплытие события?

Когда событие происходит в элементе DOM, оно затрагивает не только этот элемент. Событие «всплывает» (подобно пузырьку воздуха в воде), переходит от элемента, вызвавшего событие (event.target), к его родителю, затем поднимается еще выше, к родителю родителя элемента, пока не достигает объекта Window.  
  
Допустим, у нас есть такая разметка:

<**div** class="grandparent">

<**div** class="parent">

<**div** class="child">1</**div**>

</**div**>

</**div**>

И такой JS:

**function** **addEvent**(el, event, callback, isCapture = false) {

**if** (!el || !event || !callback || **typeof** callback !== 'function') **return**

**if** (**typeof** el === 'string') {

el = document.querySelector(el)

}

el.addEventListener(event, callback, isCapture)

}

addEvent(document, 'DOMContentLoaded', () => {

**const** child = document.querySelector('.child')

**const** parent = document.querySelector('.parent')

**const** grandparent = document.querySelector('.grandparent')

addEvent(child, 'click', **function**(e) {

console.log('child')

})

addEvent(parent, 'click', **function**(e) {

console.log('parent')

})

addEvent(grandparent, 'click', **function**(e) {

console.log('grandparent')

})

addEvent('html', 'click', **function**(e) {

console.log('html')

})

addEvent(document, 'click', **function**(e) {

console.log('document')

})

addEvent(window, 'click', **function**(e) {

console.log('window')

})

})

У метода addEventListener есть третий необязательный параметр — useCapture. Когда его значение равняется false (по умолчанию), событие начинается с фазы всплытия. Когда его значение равняется true, событие начинается с фазы погружения (для «прослушивателей» событий, прикрепленных к цели события, событие находится в целевой фазе, а не в фазах погружения или всплытия. События в целевой фазе инициируют все прослушиватели на элементе в том порядке, в котором они были зарегистрированы независимо от параметра useCapture — прим. пер.). Если мы кликнем по элементу child, в консоль будет выведено: child, parent, grandparent, html, document, window. Вот что такое всплытие события.

#### Что такое погружение события?

Когда событие происходит в элементе DOM, оно происходит не только в нем. В фазе погружения событие опускается от объекта Window до цели события через всех его предков.  
  
Разметка:

<**div** class="grandparent">

<**div** class="parent">

<**div** class="child">1</**div**>

</**div**>

</**div**>

JS:

**function** **addEvent**(el, event, callback, isCapture = false) {

**if** (!el || !event || !callback || **typeof** callback !== 'function') **return**

**if** (**typeof** el === 'string') {

el = document.querySelector(el);

}

el.addEventListener(event, callback, isCapture)

}

addEvent(document, 'DOMContentLoaded', () => {

**const** child = document.querySelector('.child')

**const** parent = document.querySelector('.parent')

**const** grandparent = document.querySelector('.grandparent')

addEvent(child, 'click', **function**(e) {

console.log('child');

}, true)

addEvent(parent, 'click', **function**(e) {

console.log('parent')

}, true)

addEvent(grandparent, 'click', **function**(e) {

console.log('grandparent')

}, true)

addEvent('html', 'click', **function**(e) {

console.log('html')

}, true)

addEvent(document, 'click', **function**(e) {

console.log('document')

}, true)

addEvent(window, 'click', **function**(e) {

console.log('window')

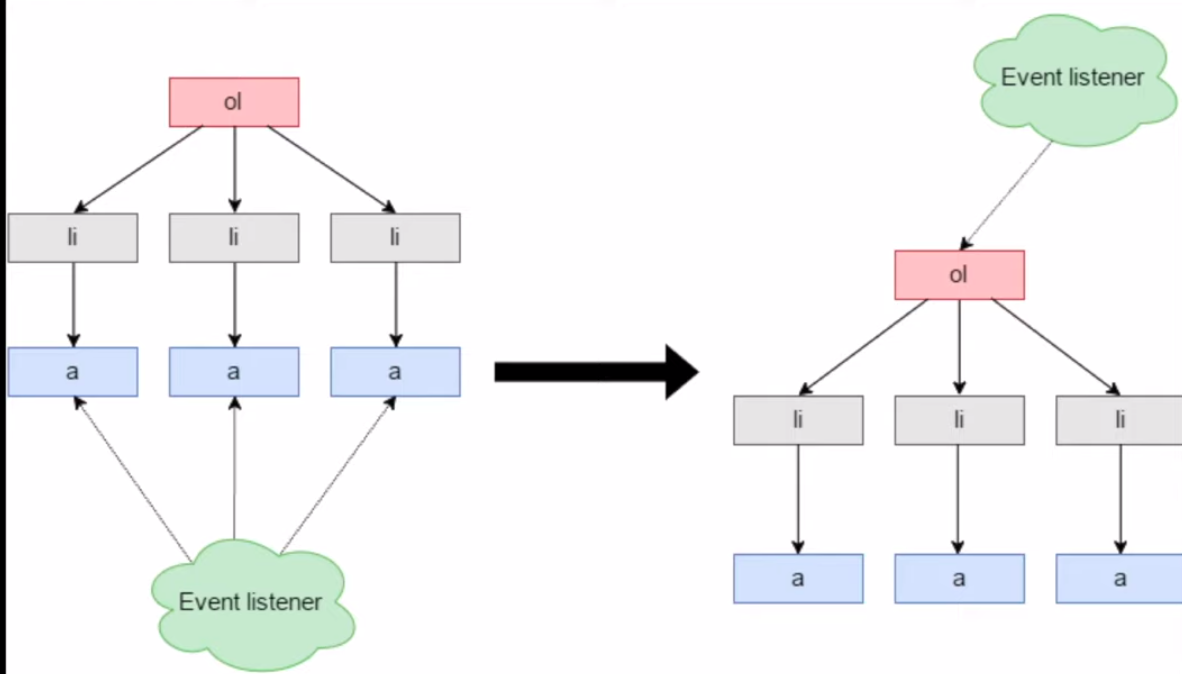
}, true)

})

У метода addEventListener есть третий необязательный параметр — useCapture. Когда его значение равняется false (по умолчанию), событие начинается с фазы всплытия. Когда его значение равняется true, событие начинается с фазы погружения. Если мы кликнем по элементу child, то увидим в консоли следующее: window, document, html, grandparent, parent, child. Это и есть погружение события.

<https://learn.javascript.ru/bubbling-and-capturing>

[**Что такое делегирование событий (Event Delegation)?**](https://youtu.be/1eIRTdgzHtw?t=576)



Делегирование событий - это приём, заключающийся в добавлении обработчиков событий к родительскому элементу, а не к дочерним элементам. Обработчик будет срабатывать всякий раз, когда событие будет запущено на дочерних элементах благодаря всплытию событий в DOM. Элемент, на котором непосредственно произошло событие, можно отслеживать при помощи event target.

https://learn.javascript.ru/event-delegation

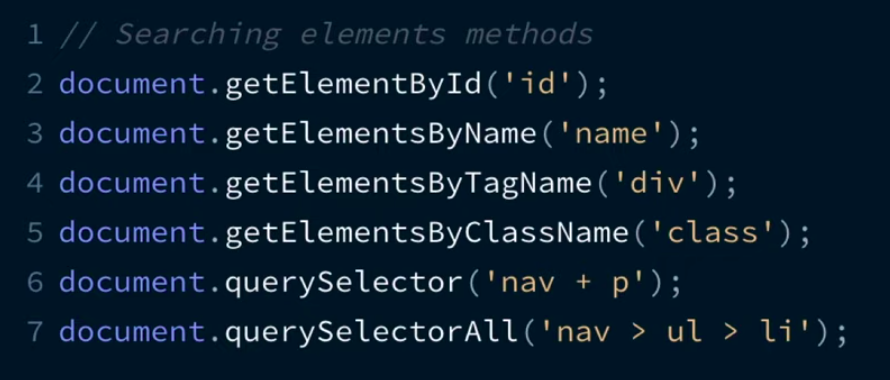
[**Разница между e.preventDefault() и e.stopPropagation()?**](https://youtu.be/CjdCxxqObaM?t=650)

Метод event.preventDefault() отключает поведение элемента по умолчанию. Если использовать этот метод в элементе form, то он предотвратит отправку формы (submit). Если использовать его в contextmenu, то контекстное меню будет отключено (данный метод часто используется в keydown для переопределения клавиатуры, например, при создании музыкального/видео плеера или текстового редактора — прим. пер.). Метод event.stopPropagation() отключает распространение события (его всплытие или погружение).



#### Как узнать об использовании метода event.preventDefault()? Для этого мы можем использовать свойство event.defaulPrevented, возвращающее логическое значение, служащее индикатором применения к элементу метода event.preventDefault.

[**Методы поиска элементов в DOM?**](https://youtu.be/CjdCxxqObaM?t=678)



https://learn.javascript.ru/searching-elements-dom

Eсли у элемента есть атрибут id, то мы можем получить его вызовом document.**getElementById(id)**, где бы он ни находился.

elem.**getElementsByTagName(tag)** ищет элементы с данным тегом и возвращает их коллекцию. Передав "\*" вместо тега, можно получить всех потомков.

elem.**getElementsByClassName(className)** возвращает элементы, которые имеют данный CSS-класс.

document.**getElementsByName(name)** возвращает элементы с заданным атрибутом name. Очень редко используется.

#### Метод elem.**querySelector(css)** возвращает первый элемент, соответствующий данному CSS-селектору.

#### Самый универсальный метод поиска – это elem.**querySelectorAll**(css), он возвращает все элементы внутри elem, удовлетворяющие данному CSS-селектору.

[**Разница между event.target и event.currentTarget?**](https://youtu.be/kx3dR6ztICU?t=539)

#### event. target – самый глубокий элемент, на котором произошло событие, который вызвал событие. event. currentTarget (= this ) – элемент, на котором в данный момент сработал обработчик (до которого «доплыло» событие) либо к которому был прикреплен прослушиватель события.

#### [**Разница между .stopPropagation() и .stopImmediatePropagation()?**](https://youtu.be/kx3dR6ztICU?t=580)

StopPropagation() отменяет дальнейшее всплытие, но на текущем элементе все обработчики срабатывают. А stopImmediatePropagation не только остановит дальнейшее всплытие, но и остановит обработку события на текущем элементе.

[**Разница между событиями load и DOMContentLoaded?**](https://youtu.be/kx3dR6ztICU?t=627)

DOMContentLoaded — событие происходит, когда браузер разобрал HTML-страницу и составил DOM-дерево; load — событие происходит, когда загрузилась и HTML страница, и все ресурсы для её отображения пользователю: стили, картинки и так далее.

[Подробнее](https://learn.javascript.ru/onload-ondomcontentloaded)

[**Разница между attribute и property у DOM-элементов?**](https://youtu.be/IooJ3P2VUYs?t=659)

Аттрибут – это статичное значение дом-элемента, которое неизменяемо и в большинстве своем может быть добавлено в hrml разметке. А вот свойство – это вычисленное значение дом-элемента. При этом его особенность в том, что оно может динамически изменяться.



https://learn.javascript.ru/attributes-and-custom-properties

[**Разница между HTMLCollection и NodeList?**](https://youtu.be/IooJ3P2VUYs?t=705)

[**HTMLCollection**](https://youtu.be/IooJ3P2VUYs?t=705) **–** это динамическая коллекция, которая представляет собой массивоподобный и итерируемый объект дочерних элементов. NodeList – статический список нод и узлов, в который входят все найденные в документе элементы,

[](https://doka.guide/js/htmlcollection-and-nodelist/" \l ":~:text=HTMLCollection и NodeList — это очень,них нет привычных методов массива.)Подробнее

[**Как динамически добавить элемент на HTML-страницу?**](https://youtu.be/nvktMVFM0_M?t=551)

https://learn.javascript.ru/modifying-document

[**Типы узлов DOM-дерева?**](https://youtu.be/7TvS0iKR3_c?t=201)

Существует 12 типов узлов. Но на практике мы в основном работаем с 4 из них:

* document – «входная точка» в DOM.
* узлы-элементы – HTML-теги, основные строительные блоки.
* текстовые узлы – содержат текст.
* комментарии – иногда в них можно включить информацию, которая не будет показана, но доступна в DOM для чтения JS.

https://learn.javascript.ru/dom-nodes

https://habr.com/ru/companies/ruvds/articles/539096/

**Свойства для перемещения по DOM-дереву?**

https://learn.javascript.ru/dom-navigation

https://www.w3schools.com/js/js\_htmldom\_navigation.asp

**Виды событий в JavaScript?**

Все события JavaScript можно разделить на следующие категории: события мыши (Mouse Events), события клавиатуры (Keyboard Events), события объектов и фреймов (Frame/Object Events), события формы и элементов управления (Form Events), события перетаскивания (Drag Events), события анимации (Animation Events), события буфера обмена (Clipboard Events), события мультимедиа (Media Events), события трансформации (Transition Events), события, посылаемые сервером (Server-Sent Events), события касания (Touch Events), события печати (Print Events), разные события (Misc Events).

[Подробнее](https://learn.javascript.ru/introduction-browser-events)

[**Как добавить обработчик события на DOM-элемент?**](https://youtu.be/7TvS0iKR3_c?t=425)

[Подробнее](https://learn.javascript.ru/introduction-browser-events)

https://www.w3schools.com/js/js\_htmldom\_events.asp

[**Как удалить обработчик события с DOM-элемента?**](https://youtu.be/7TvS0iKR3_c?t=505)

removeEventListener() Удаляет обработчик события, который был зарегистрирован при помощи EventTarget.

[**Сколько аргументов принимает addEventListener?**](https://youtu.be/7TvS0iKR3_c?t=538)

https://www.w3schools.com/jsref/met\_element\_addeventlistener.asp

<https://developer.mozilla.org/ru/docs/Web/API/EventTarget/addEventListener>

Данный метод принимает три аргумента: тип события, исполняемую функцию и булево значение (значение логического типа). Первый атрибут — это имя события. Второй — функция, которая будет исполняться при наступлении события. Это может быть ссылка на функцию (как в нашем случае) или просто анонимная функция. Третий атрибут, принимающий значение true или false, указывает, каким образом срабатывают множественные события. Например, мы прослушиваем событие click на двух вложенных элементах (один элемент вложен в другой). Когда пользователь щелкает на этих элементах, два события click срабатывают в порядке, определяемом этим атрибутом. Если его значение для одного из элементов равно true, то считается, что событие для этого элемента произошло первым, а для другого элемента — вторым по счету. Обычно для большинства ситуаций хватает значения false.

[**Разница между innerHTML и outerHTML?**](https://youtu.be/xZLxdts7ZW4?t=621)

Свойство outerHTML содержит HTML элемента целиком. Это как innerHTML плюс сам элемент. Будьте осторожны: в отличие от innerHTML , запись в outerHTML не изменяет элемент. Вместо этого элемент заменяется целиком во внешнем контексте.

https://www.makeuseof.com/html-inner-outer-dom-difference/

The innerHTML property is part of the DOM and you can access it via JavaScript. You can use it to get or set the contents of an element. This content excludes the element's own tag and only includes the markup that represents the element's children, in the form of a string.

The outerHTML property is like innerHTML in many ways. You can use it to get or set the contents of a selected element. However, it also sets the markup representing the element itself. This includes the opening tag, any properties, and—where relevant—the closing tag.

The innerHTML and outerHTML DOM properties have many similarities, but one key difference. The innerHTML property captures the HTML contents of an element. In contrast, the outerHTML property captures the HTML that represents the element itself and its content.

You can use these properties to read and write HTML content via the DOM. The DOM will be a common, important concept throughout your JavaScript development process.

[**Расскажите про координаты в браузере?**](https://youtu.be/70VnuTXi4Wk?t=644)

Система координат в браузере аналогична плоской системе координат из школьной алгебры. Есть две оси: ось y и ось x, на этой плоскости располагаются все элементы сайта. Единственным отличием от классической системы координат является то, что положительное направление оси y в браузере повёрнуто вниз.

<https://learn.javascript.ru/coordinates> [Дока](https://doka.guide/js/coordinates/" \l ":~:text=Система координат в браузере аналогична,y в браузере повёрнуто вниз.)

[**Разница между JSON и XML?**](https://youtu.be/XtQPrt8G0n8?t=28)

Формат JSON используется для хранения и передачи данных, а XML - для представления данных в машиночитаемом виде. JSON набирает популярность как средство хранения данных для веб-приложений благодаря своей простоте. В отличие от него, XML все еще используется для передачи структурированных данных через Интернет.

https://coderlessons.com/tutorials/veb-razrabotka/arkhitektura-veb-servisov/10-json-protiv-xml

[**Для чего используется свойство window.navigator?**](https://youtu.be/lZNWrW39ELM?t=493)

Свойство window.navigator возвращает объект описания приложения (user agent), которое выполняет скрипт. В подавляющем большинстве случаев это приложение — браузер. Объект содержит свойства, описывающие браузер, и методы для выполнения действий.

Часто используемые свойства:

userAgent возвращает строку, которая содержит название браузера. Не стоит использовать это свойство, чтобы определить браузер пользователя! Спецификация рекомендует браузерам передавать минимум информации в userAgent, значение может меняться от версии к версии.

language возвращает предпочитаемый язык интерфейса в виде языкового тега. Например, en, ru, en-US и т.д. Обычно это язык, установленный в настройках браузера.

languages возвращает массив предпочитаемых языков в порядке предпочтительности. Первый в списке будет язык, который возвращает navigator.language.

cookieEnabled возвращает true, если браузер пользователя поддерживает куки и они включены, false в противном случае.

onLine возвращает true, если у пользователя есть подключение к сети. Браузеры вкладывают разные смыслы в понятие «онлайн», поэтому это свойство — ненадёжный источник данных.

Объект navigator содержит множество других свойств, большинство из них экспериментальные или поддерживаются конкретными браузерами.

Методы объекта navigator служат для взаимодействия с другими WebAPI. Например, метод vibrate, который вызывает вибрацию пользовательского устройства, если она поддерживается: navigator.vibrate(200)

[**Для чего используется метод .focus()?**](https://youtu.be/lZNWrW39ELM?t=551)

Метод focus устанавливает фокус на элементе (чаще всего на инпуте). Это значит, что в этом инпуте начнет моргать курсор и вводимый с клавиатуры текст будет попадать именно в этот инпут.

[ДОКА](https://doka.guide/js/element-focus/)

[**Для чего используется свойство .forms?**](https://youtu.be/lZNWrW39ELM?t=616)

Свойство Form используется для ссылки на форму или для ссылки на форму, связанную с вложенным элементом управления.

https://learn.javascript.ru/form-elements

https://www.w3schools.com/js/js\_validation.asp

[**Для чего используется метод .scrollIntoView()?**](https://youtu.be/lZNWrW39ELM?t=682)

Метод scrollIntoView() позволяет программно прокрутить окно до определённого элемента.

[ДОКА](https://doka.guide/js/element-scrollintoview/" \l ":~:text=Метод scrollIntoView() позволяет программно прокрутить окно до определённого элемента.)

[**Как использовать media выражения в JavaScript?**](https://youtu.be/3kvKFfPteFg?t=30)

[Ответ](https://webdevblog.ru/rabotaem-s-media-zaprosami-cherez-javascript/)

[**Разница между методами .submit() и .requestSubmit()?**](https://youtu.be/3kvKFfPteFg?t=90)

As you can see the name input has a required attribute, the form will however still be submitted, because submit() doesn't care about validation.

This is where requestSubmit() is useful. It acts exactly like a trigger on a submit button. If you use form.requestSubmit() instead of form.submit() the form will only be submitted if it is valid. If you leave the name input empty the browser will show a hint to the user that they have to fill out that field and the form will not be submitted.

Browser support is pretty good, except for Internet Explorer and more importantly Safari. So, you should make a feature detection before using it.

**Расскажите о IntersectionObserver?**

Intersection Observer — браузерный API, который позволяет асинхронно отслеживать пересечение элемента с его родителем или областью видимости документа (viewport). В момент пересечения можно запустить какое-либо действие, например, подгрузить дополнительные посты в ленте новостей («бесконечный скролл») или сделать «ленивую» загрузку контента.

[ДОКА](https://doka.guide/js/intersection-observer/)

[**Расскажите о URLSearchParams?**](https://youtu.be/hL5yFo9Pms4?t=98)

URLSearchParams — это класс, предоставляющий удобное API для формирования строки поисковых параметров, которую потом можно использовать для формирования полного адреса. Все параметры в строке будут закодированы для безопасной вставки в адрес. Также этот класс можно встретить как часть класса URL.

[ДОКА](https://doka.guide/js/urlsearchparams/" \l ":~:text=URLSearchParams — это класс%2C предоставляющий удобное,встретить как часть класса URL .)

https://developer.mozilla.org/ru/docs/Web/API/URLSearchParams

[**Какие есть ограничения у window.close()?**](https://youtu.be/hL5yFo9Pms4?t=149)

[Ответ](https://habr.com/ru/companies/ruvds/articles/550194/)

### [**Async JS**](https://youtu.be/kx3dR6ztICU?t=681)