Создание элемента

**document.createElement(tag)**

Создаёт новый элемент с заданным тегом:



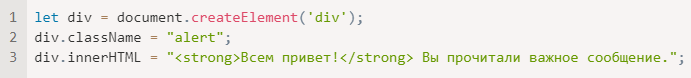
**document.createTextNode(text)**

Создаёт новый *текстовый узел* с заданным текстом:



**Заполнение элемента**

В нашем случае сообщение – это div с классом alert и HTML в нём:



**[Методы вставки](https://learn.javascript.ru/modifying-document" \l "metody-vstavki)**

Вот методы для различных вариантов вставки:

* 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.

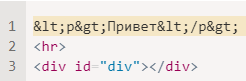
Эти методы могут вставлять несколько узлов и текстовых фрагментов за один вызов.

Например, здесь вставляется строка и элемент:



Весь текст вставляется *как текст*.

Поэтому финальный HTML будет:



Другими словами, строки вставляются безопасным способом, как делает это elem.textContent.

Поэтому эти методы могут использоваться только для вставки DOM-узлов или текстовых фрагментов.

## [insertAdjacentHTML/Text/Element](https://learn.javascript.ru/modifying-document" \l "insertadjacenthtml-text-element)

elem.insertAdjacentHTML(where, html)

Первый параметр – это специальное слово, указывающее, куда по отношению к elem производить вставку. Значение должно быть одним из следующих:

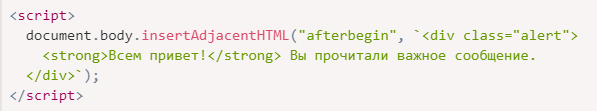
* "beforebegin" – вставить html непосредственно перед elem,
* "afterbegin" – вставить html в начало elem,
* "beforeend" – вставить html в конец elem,
* "afterend" – вставить html непосредственно после elem.

Второй параметр – это HTML-строка, которая будет вставлена именно «как HTML».

У метода есть два брата:

* elem.insertAdjacentText(where, text) – такой же синтаксис, но строка text вставляется «как текст», вместо HTML,
* elem.insertAdjacentElement(where, elem) – такой же синтаксис, но вставляет элемент elem.

Они существуют, в основном, чтобы унифицировать синтаксис. 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.



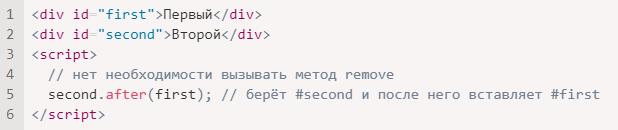
## [Удаление узлов](https://learn.javascript.ru/modifying-document" \l "udalenie-uzlov)

node.remove()

Например, сделаем так, чтобы наше сообщение удалялось через секунду:



**Все методы вставки автоматически удаляют узлы со старых мест.**



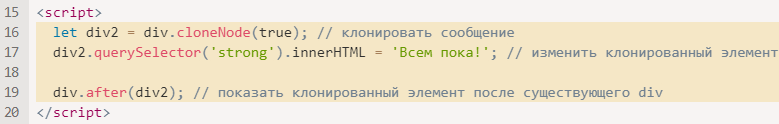
## [Клонирование узлов: cloneNode](https://learn.javascript.ru/modifying-document" \l "klonirovanie-uzlov-clonenode)

Как вставить ещё одно подобное сообщение?

Мы могли бы создать функцию и поместить код туда. Альтернатива – *клонировать* существующий div и изменить текст внутри него (при необходимости).

Иногда, когда у нас есть большой элемент, это может быть быстрее и проще.

* Вызов elem.cloneNode(true) создаёт «глубокий» клон элемента – со всеми атрибутами и дочерними элементами. Если мы вызовем elem.cloneNode(false), тогда клон будет без дочерних элементов.



## [DocumentFragment](https://learn.javascript.ru/modifying-document" \l "document-fragment)

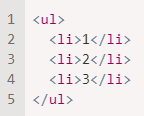
DocumentFragment является специальным DOM-узлом, который служит обёрткой для передачи списков узлов.

Мы можем добавить к нему другие узлы, но когда мы вставляем его куда-то, он «исчезает», вместо него вставляется его содержимое.

Например, getListContent ниже генерирует фрагмент с элементами <li>, которые позже вставляются в <ul>:



Обратите внимание, что на последней строке с (\*) мы добавляем DocumentFragment, но он «исчезает», поэтому структура будет:



DocumentFragment редко используется. Зачем добавлять элементы в специальный вид узла, если вместо этого мы можем вернуть массив узлов? Переписанный пример:



Мы упоминаем DocumentFragment в основном потому, что он используется в некоторых других областях, например, для элемента [template](https://learn.javascript.ru/template-element), который мы рассмотрим гораздо позже.

## [Устаревшие методы вставки/удаления](https://learn.javascript.ru/modifying-document" \l "ustarevshie-metody-vstavki-udaleniya)

Есть несколько других, более старых, методов вставки и удаления, которые существуют по историческим причинам.

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).

Все эти методы возвращают вставленный/удалённый узел. Другими словами, parentElem.appendChild(node) вернёт node. Но обычно возвращаемое значение не используют, просто вызывают метод.

## [Несколько слов о «document.write»](https://learn.javascript.ru/modifying-document" \l "neskolko-slov-o-document-write)

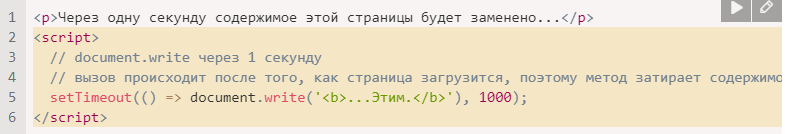
Вызов document.write(html) записывает html на страницу «прямо здесь и сейчас». Строка html может быть динамически сгенерирована, поэтому метод достаточно гибкий. Мы можем использовать JavaScript, чтобы создать полноценную веб-страницу и записать её в документ.

Этот метод пришёл к нам со времён, когда ещё не было ни DOM, ни стандартов… Действительно старые времена. Он всё ещё живёт, потому что есть скрипты, которые используют его.

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.



Так что после того, как страница загружена, он уже непригоден к использованию, в отличие от других методов DOM, которые мы рассмотрели выше.

Это его недостаток.

Есть и преимущество. Технически, когда document.write запускается во время чтения HTML браузером, и что-то пишет в документ, то браузер воспринимает это так, как будто это изначально было частью загруженного HTML-документа.

Поэтому он работает невероятно быстро, ведь при этом *нет модификации DOM*. Метод пишет прямо в текст страницы, пока DOM ещё в процессе создания.

Так что, если нам нужно динамически добавить много текста в HTML, и мы находимся на стадии загрузки, и для нас очень важна скорость, это может помочь. Но на практике эти требования редко сочетаются. И обычно мы можем увидеть этот метод в скриптах просто потому, что они старые.

**[Итого](https://learn.javascript.ru/modifying-document" \l "itogo)**

* Методы для создания узлов:
  + document.createElement(tag) – создаёт элемент с заданным тегом,
  + document.createTextNode(value) – создаёт текстовый узел (редко используется),
  + elem.cloneNode(deep) – клонирует элемент, если deep==true, то со всеми дочерними элементами.
* Вставка и удаление:
  + node.append(...nodes or strings) – вставляет в node в конец,
  + node.prepend(...nodes or strings) – вставляет в node в начало,
  + node.before(...nodes or strings) – вставляет прямо перед node,
  + node.after(...nodes or strings) – вставляет сразу после node,
  + node.replaceWith(...nodes or strings) – заменяет node.
  + node.remove() – удаляет node.
* Устаревшие методы:
  + parent.appendChild(node)
  + parent.insertBefore(node, nextSibling)
  + parent.removeChild(node)
  + parent.replaceChild(newElem, node)

Все эти методы возвращают node.

* Если нужно вставить фрагмент HTML, то elem.insertAdjacentHTML(where, html) вставляет в зависимости от where:
  + "beforebegin" – вставляет html прямо перед elem,
  + "afterbegin" – вставляет html в elem в начало,
  + "beforeend" – вставляет html в elem в конец,
  + "afterend" – вставляет html сразу после elem.

Также существуют похожие методы elem.insertAdjacentText и elem.insertAdjacentElement, они вставляют текстовые строки и элементы, но они редко используются.

* Чтобы добавить HTML на страницу до завершения её загрузки:
  + document.write(html)

После загрузки страницы такой вызов затирает документ. В основном встречается в старых скриптах.