

## **ПМЗ Разработка модулей ПО.**

**РО 3.2 Разрабатывать модули с применением DOM API,  
Regexp, HTTP**

# **Тема 1. JS: DOM API.**

## **Лекция 29. Манипуляции с DOM.**

# **Цель занятия:**

**Познакомиться с методами создания, вставки, удаления, перемещения и клонирования элементов в DOM.  
Научиться использовать их для динамического изменения структуры страницы.**

# **Учебные вопросы:**

- 1. Создание элементов.**
- 2. Вставка элементов.**
- 3. Удаление и перемещение.**
- 4. Клонирование узлов.**
- 5. Практика.**

# 1. Создание элементов.

## ◆ 1. Создание элемента

Новый элемент создаётся с помощью метода:

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

- В скобках указывается имя тега.
- Пока элемент существует только в памяти и не отображается на странице.
- Чтобы его увидеть, нужно вставить в DOM (append, prepend и др.).

## ◆ 2. Создание текстового узла.

Иногда нужен не элемент, а текстовый узел:

```
let text = document.createTextNode("Привет, мир!");
```

- Такой узел можно вставить внутрь элемента.
- Но чаще используют свойство `textContent` у элемента (см. ниже).

### ◆ 3. Установка текста.

После создания элемента можно добавить текст:

```
let p = document.createElement("p");  
p.textContent = "Это новый абзац!";
```

⚠ Отличие от innerHTML:

- textContent безопаснее (не обрабатывает HTML-теги).
- innerHTML вставляет HTML-код, что может быть небезопасно при вводе от пользователя.

## ◆ 4. Установка атрибутов.

У элементов можно задавать атрибуты разными способами:

Через `setAttribute`:

```
let img = document.createElement("img");  
img.setAttribute("src", "cat.png");  
img.setAttribute("alt", "Милый котик");
```

Через свойства:

```
img.src = "cat.png";  
img.alt = "Рыжий котик";
```

Оба варианта работают, но свойства предпочтительнее, когда они есть (например, `img.src`).



## ◆ 5. Комбинированный пример

```
let link = document.createElement("a");      // создаём <a>
link.textContent = "Перейти";                // задаём текст
link.setAttribute("href", "https://example.com"); // задаём атрибут
document.body.append(link);                  // вставляем на страницу
```

Результат в HTML:

```
<a href="https://example.com">Перейти</a>
```

## Выводы:

- Элементы создаются через `document.createElement`.
- Текст можно добавить `createTextNode` или `textContent`.
- Атрибуты устанавливаются через `setAttribute` или свойства элемента.
- Пока элемент не вставлен в DOM, он существует только в памяти.

## 2. Вставка элементов.

После того как элемент создан (например, через `document.createElement`), его нужно вставить в дерево документа, чтобы он отобразился на странице.

## ◆ 1. append и prepend.

Эти методы вставляют узлы внутрь элемента:

```
let list = document.querySelector("ul");  
let li = document.createElement("li");  
li.textContent = "Новый пункт";  
// Добавить в конец списка  
list.append(li);  
// Добавить в начало списка  
list.prepend(li);
```



### Особенности:

- Могут вставлять сразу несколько узлов или строки текста.
- Элементы, которые вставляются повторно, перемещаются, а не копируются.

## ◆ 2. before и after.

Эти методы вставляют узлы рядом с элементом, снаружи:

```
let p = document.querySelector("p");  
let note = document.createElement("div");  
note.textContent = "Примечание";  
// Вставить перед <p>  
p.before(note);  
// Вставить после <p>  
p.after(note);
```

### ◆ 3. Старый синтаксис: appendChild и insertBefore.

Раньше использовались более «низкоуровневые» методы:

```
let ul = document.querySelector("ul");
let li = document.createElement("li");
li.textContent = "Ещё один пункт";
// В конец
ul.appendChild(li);
// Вставить перед первым элементом
ul.insertBefore(li, ul.firstChild);
```

#### Особенности:

- Работают только с узлами (нельзя передавать текстовые строки напрямую).
- Более многословны, но до сих пор используются в старом коде.

## ◆ 4. Пример со всеми методами:

```
<ul id="list">  
  <li>Первый</li>  
  <li>Второй</li>  
</ul>
```

```
let ul = document.getElementById("list");  
  
// Создаём элементы  
let li1 = document.createElement("li");  
li1.textContent = "Append (в конец)";  
ul.append(li1);
```

```
let li2 = document.createElement("li");  
li2.textContent = "Prepend (в начало)";  
ul.prepend(li2);
```

```
let li3 = document.createElement("li");  
li3.textContent = "Before (перед списком)";  
ul.before(li3);
```

```
let li4 = document.createElement("li");  
li4.textContent = "After (после списка)";  
ul.after(li4);
```



# Сравнение методов вставки элементов

Метод	Куда вставляет	Принимает	Особенности
append	В конец выбранного элемента	Узлы + строки	Можно сразу несколько аргументов
prepend	В начало выбранного элемента	Узлы + строки	Можно сразу несколько аргументов
before	Перед выбранным элементом (снаружи)	Узлы + строки	Вставка вне родителя
after	После выбранного элемента (снаружи)	Узлы + строки	Вставка вне родителя
appendChild	В конец выбранного элемента	Только узел	Старый метод, 1 аргумент
insertBefore	Перед указанным потомком	Только узел	Нужно явно указать «перед кем»

## Выводы:

- `append` / `prepend` → вставка внутрь элемента.
- `before` / `after` → вставка рядом с элементом.
- `appendChild` / `insertBefore` → старый синтаксис, работает только с узлами.
- При повторной вставке элемент перемещается, а не копируется.

# 3. Удаление и перемещение.

## ◆ 1. Удаление элементов.

Современный способ — метод `remove()`:

```
let p = document.querySelector("p");  
p.remove(); // элемент исчезнет из DOM
```

⚠ Поддержка есть во всех современных браузерах.

## Старый способ — removeChild:

```
let ul = document.querySelector("ul");  
let li = ul.firstElementChild;  
ul.removeChild(li);
```

- Используется реже, в основном для совместимости со старым кодом.
- Нужно указывать родителя.

## ◆ 2. Перемещение элементов.

В DOM нельзя иметь один и тот же узел в двух местах одновременно.

Если элемент вставить снова — он переместится, а не скопируется.

## 👉 Пример:

```
<ul id="list1">  
  <li id="task">Задача</li>  
</ul>  
  
<ul id="list2"></ul>
```

```
let task = document.getElementById("task");  
let list2 = document.getElementById("list2");  
// Перемещаем <li> из list1 в list2  
list2.append(task);
```

Теперь элемент `<li>` будет в `list2`, а в `list1` его уже нет.

## Выводы:

- `remove()` — современный способ удаления элементов.
- `removeChild()` — устаревший, но может встречаться в старом коде.
- Повторная вставка существующего элемента приводит к его перемещению, а не копированию.

## 4. Клонирование узлов.

### ◆ 1. Метод cloneNode

Любой DOM-узел можно скопировать с помощью метода:

```
let clone = element.cloneNode(deep);
```

Аргумент `deep` принимает значение:

- `false` → поверхностная копия (только сам элемент, без содержимого).
- `true` → глубокая копия (элемент со всем его содержимым).



## ◆ 2. Поверхностное копирование.

```
<div id="box">  
  <p>Текст</p>  
</div>
```

```
let box = document.getElementById("box");  
let clone = box.cloneNode(false); // копия только <div>  
document.body.append(clone);
```



В результате появится пустой <div></div>.

### ◆ 3. Глубокое копирование.

```
let box = document.getElementById("box");  
let clone = box.cloneNode(true); // копия вместе с потомками  
document.body.append(clone);
```

 В результате появится копия всего блока вместе с  
<p>Текст</p>.

## 👉 Выводы:

- `cloneNode(false)` → копируется только сам элемент.
- `cloneNode(true)` → копируется элемент и всё его содержимое.
- Клон не вставляется автоматически в DOM, его нужно добавить вручную (`append`, `prepend`, `before`, `after`).

# 5. Практика: динамически обновляемый список задач (Todo-list)

## ◆ 1. Исходный шаблон:

```
<h2>Список задач</h2>
<input id="taskInput" type="text" placeholder="Новая задача">
<button id="addBtn">Добавить</button>

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

## ◆ 2. Создание <li> по кнопке «Добавить»

```
let input = document.getElementById("taskInput");
let addBtn = document.getElementById("addBtn");
let list = document.getElementById("tasks");

addBtn.onclick = () => {
  if (input.value.trim() !== "") {
    let li = document.createElement("li");
    li.textContent = input.value;
    list.append(li);
    input.value = ""; // очистка
  }
};
```

### ◆ 3. Удаление задачи

```
let li = document.createElement("li");  
li.textContent = input.value;  
  
let delBtn = document.createElement("button");  
delBtn.textContent = "X";  
delBtn.onclick = () => li.remove();  
  
li.append(" ", delBtn);  
list.append(li);
```

#### ◆ 4. Перемещение задачи в конец списка

```
let cloneBtn = document.createElement("button");
cloneBtn.textContent = "📄";
cloneBtn.onclick = () => {
  let copy = li.cloneNode(true);
  list.append(copy);
};

li.append(" ", cloneBtn);
```

## ◆ 5. Клонирование задачи




```
let cloneBtn = document.createElement("button");
cloneBtn.textContent = "📄";
cloneBtn.onclick = () => {
  let copy = li.cloneNode(true);
  list.append(copy);
};

li.append(" ", cloneBtn);
```



## Что в итоге

Каждая задача в списке имеет:

-  кнопку для удаления,
-  кнопку для перемещения в конец,
-  кнопку для клонирования.

# Контрольные вопросы:

- Как создать новый элемент в DOM?
- Чем отличаются append и prepend от before и after?
- Что происходит при повторной вставке одного и того же элемента?
- Чем отличается remove() от removeChild()?
- Что делает cloneNode(true) и зачем может понадобиться глубокое копирование?

# Домашнее задание:

1. <https://ru.hexlet.io/courses/js-dom>

## 9 Манипулирование DOM-деревом

Учимся менять DOM-дерево, добавлять и удалять элементы

## 10 Управление узлами DOM

Учимся модифицировать элементы, разбираем разницу между атрибутами и свойствами

# **Материалы лекций:**

<https://github.com/ShViktor72/Education2025>

# **Обратная связь:**

[colledge20education23@gmail.com](mailto:colledge20education23@gmail.com)