**Конспект «Манипуляции с DOM»**

**Коллекции**

Есть несколько способов, чтобы найти сразу несколько элементов на странице. Элементы записываются в структуру, которая называется *коллекцией*. Коллекции похожи на массив, но ими не являются. При этом к элементам коллекции можно обращаться по индексу и перебирать в цикле for, как и обычные массивы.

Ниже несколько способов получения коллекций:

* **element.querySelectorAll()** — возвращает **все** элементы, которые подходят под указанное правило. Эта запись остаётся статичной и изменения в DOM на неё никак не влияют. Можно сказать, что querySelectorAll работает, как любая переменная, в которую мы записали какое-нибудь значение. Пока мы не переопределим переменную, в ней так и будет находиться то значение, которое мы в неё записали, независимо от того, что происходит в коде. Поэтому такая коллекция называется статичной.
* **parentElement.children** — вызывается на родительском элементе и собирает все дочерние элементы в *динамическую коллекцию*. Такие коллекции реагируют на изменения в DOM. Если один из элементов коллекции будет удалён из DOM, то он пропадёт и из коллекции.

**Работа с элементами**

Удаление элемента

Удалять элементы со страницы можно разными способами, один из самых простых — вызов метода remove на элементе, который нужно удалить.

element.remove();

Метод из примера выше удалит element из DOM.

Клонирование элемента

С помощью клонирования мы можем копировать элементы сколько угодно раз и вставлять их в любые места на странице. Для этого существует метод cloneNode. Синтаксис такой:

element.cloneNode(true);

// Вернёт склонированный элемент со всеми вложенностями

element.cloneNode(false);

// Вернёт склонированный элемент без вложенностей

element.cloneNode();

// 0\_o

При передаче true в качестве аргумента клонируется сам элемент вместе со всеми вложенностями. Причём клонируются атрибуты, классы и текстовое содержимое всех вложенностей. Такое клонирование называется глубоким.

Если передать методу в качестве аргумента значение false, то будет скопирован сам элемент со своими классами и атрибутами, но без дочерних элементов.

Лучше всегда явно передавать аргумент в cloneNode, чтобы избежать ошибок в работе программ.

Как получить текст из поля ввода

Нужно обратиться к свойству поля ввода value. Оно хранит информацию, введённую в поле.

input.value;

Результат можно сохранить в переменную и использовать дальше в коде.

**Шаблоны и тег template**

Тег template хранит в себе шаблон для будущих элементов. Он там же, где и вся разметка сайта, только его содержимое не отображается на странице.

Чтобы получить template в JavaScript, можно найти его по идентификатору. Это уникальное название записывают в атрибут id. Такой атрибут можно указывать для разных элементов, главное соблюдать правило — значение атрибута не должно повторяться на одной странице.

Шаблон в разметке:

<body>

…

<template id="text-template">

<p class="text"></p>

</template>

</body>

Поиск элемента в JavaScript:

document.querySelector('#text-template');

Решётка в параметре querySelector обозначает, что искать надо по id.

Внутри template находится document-fragment или просто фрагмент. Он является хранилищем содержимого тега template. Именно благодаря ему разметка из template не отображается на странице.

Чтобы получить необходимые элементы в шаблоне, надо обратиться к document-fragment, он находится в свойстве content и дальше искать нужные элементы привычными методами поиска.

<body>

…

<template id="text-template">

<p class="text"></p>

</template>

</body>

Если мы хотим найти элемент в шаблоне, надо искать так:

var template = document.querySelector('#text-template');

// Нашли template в документе

var content = template.content;

// Получили содержимое, фрагмент

var text = content.querySelector('.text');

// Нашли нужный шаблон

Эту запись можно сократить. Например, записать в отдельную переменную контент, а в другую искомый шаблон.

var textTemplate = document.querySelector('#text-template').content;

var text = textTemplate.querySelector('.text');

**События**

* **change** — срабатывает, когда состояние поля ввода меняется. Например, невыбранный чекбокс становится выбранным или наоборот.
* **submit** — реагирует на отправку формы. Формы отправляются по умолчанию так же, как при клике по ссылке происходит переход по указанному адресу. Если вам не нужно отправлять форму в каких-то случаях, отмените действие по умолчанию с помощью preventDefault.

<ul class="todo-list">

<li class="todo-list-item">

<label>

<input class="todo-list-input" type="checkbox">

<span>Покормить кота</span>

</label>

</li>

<li class="todo-list-item">

<label>

<input class="todo-list-input" type="checkbox">

<span>Уехать из страны</span>

</label>

</li>

<li class="todo-list-item">

<label>

<input class="todo-list-input" type="checkbox">

<span>Вдариться в бега</span>

</label>

</li>

</ul>

<p class="empty-tasks hidden">Все задачи выполнены. Новых задач нет.</p>

<form class="add-form" action="https://echo.htmlacademy.ru/courses" method="post">

<input class="add-form-input" type="text" aria-label="Описание задачи" placeholder="Например, купить слона" required>

<button class="add-form-button" type="submit">Добавить задачу</button>

</form>

<template id="task-template">

<li class="todo-list-item">

<label>

<input type="checkbox" class="todo-list-input">

<span></span>

</label>

</li>

</template>

var list = document.querySelector('.todo-list');

var items = list.children;

var emptyListMessage = document.querySelector('.empty-tasks');

var newItemForm = document.querySelector('.add-form');

var newItemTitle = newItemForm.querySelector('.add-form-input');

var taskTemplate = document.querySelector('#task-template').content;

var newItemTemplate = taskTemplate.querySelector('.todo-list-item');

var toggleEmptyListMessage = function () {

if (items.length === 0) {

emptyListMessage.classList.remove('hidden');

} else {

emptyListMessage.classList.add('hidden');

}

};

var addCheckHandler = function (item) {

var checkbox = item.querySelector('.todo-list-input');

checkbox.addEventListener('change', function () {

item.remove();

toggleEmptyListMessage();

});

};

for (var i = 0; i < items.length; i++) {

addCheckHandler(items[i]);

}

newItemForm.addEventListener('submit', function (evt) {

evt.preventDefault();

var taskText = newItemTitle.value;

var task = newItemTemplate.cloneNode(true);

var taskDescription = task.querySelector('span');

taskDescription.textContent = taskText;

addCheckHandler(task);

list.appendChild(task);

toggleEmptyListMessage();

newItemTitle.value = '';

});