



Лекция №6

DOM и События





Изменения в Дом



Создание элемента

■ DOM-узел можно создать двумя методами:

`document.createElement(tag)`

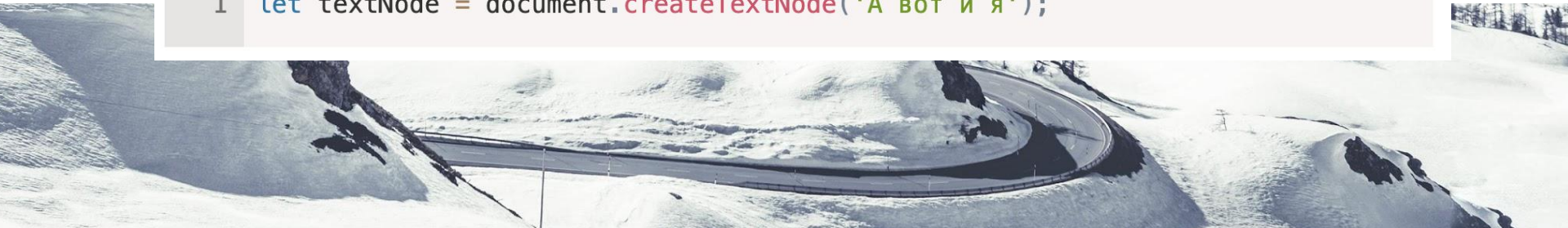
Создаёт новый элемент с заданным тегом:

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

`document.createTextNode(text)`

Создаёт новый текстовый узел с заданным текстом:

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



Создание сообщения

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

```
1 let div = document.createElement('div');  
2 div.className = "alert";  
3 div.innerHTML = "<strong>Всем привет!</strong> Вы прочитали важное сообщение."
```

Мы создали элемент, но пока он только в переменной. Мы не можем видеть его на странице, поскольку он не является частью документа.






Методы вставки

Чтобы наш `div` появился, нам нужно вставить его где-нибудь в `document`. Например, в `document.body`.

Для этого есть метод `append`, в нашем случае: `document.body.append(div)`.

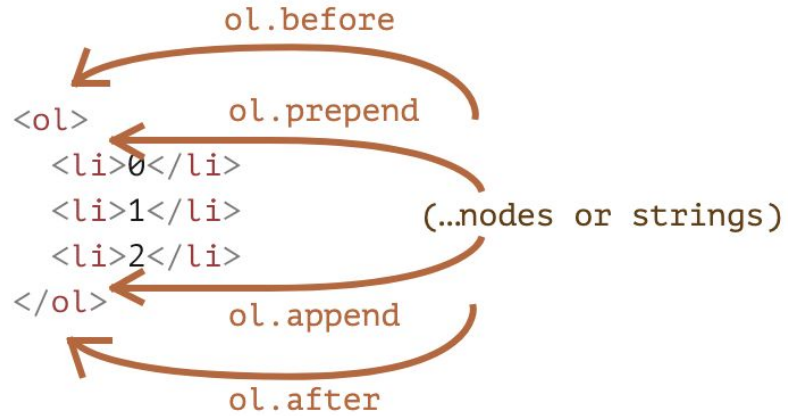




Вот методы для различных вариантов вставки:

- `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.







Удаление узлов

Для удаления узла есть методы `node.remove()`.



Клонирование узлов: `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.

Пример копирования сообщения:



Стили



JavaScript может менять и классы, и свойство `style`.

Классы – всегда предпочтительный вариант по сравнению со `style`. Мы должны манипулировать свойством `style` только в том случае, если классы «не могут справиться».

Например, использование `style` является приемлемым, если мы вычисляем координаты элемента динамически и хотим установить их из JavaScript:

```
1 let top = /* сложные расчёты */;  
2 let left = /* сложные расчёты */;  
3  
4 elem.style.left = left; // например, '123px', значение вычисляется во время р  
5 elem.style.top = top; // например, '456px'
```



className и classList

Изменение класса является одним из наиболее часто используемых действий в скриптах.

Когда-то давно в JavaScript существовало ограничение: зарезервированное слово типа `"class"` не могло быть свойством объекта. Это ограничение сейчас отсутствует, но в то время было невозможно иметь свойство `elem.class`.

Поэтому для классов было введено схожее свойство `"className"`: `elem.className` соответствует атрибуту `"class"`.



Для этого есть другое свойство: `elem.classList`.

`elem.classList` – это специальный объект с методами для добавления/удаления одного класса.

Например:

```
1 <body class="main page">
2   <script>
3     // добавление класса
4     document.body.classList.add('article');
5
6     alert(document.body.className); // main page article
7   </script>
8 </body>
```



Так что мы можем работать как со строкой полного класса, используя `className`, так и с отдельными классами, используя `classList`. Выбираем тот вариант, который нам удобнее.

Методы `classList`:

- `elem.classList.add/remove("class")` – добавить/удалить класс.
- `elem.classList.toggle("class")` – добавить класс, если его нет, иначе удалить.
- `elem.classList.contains("class")` – проверка наличия класса, возвращает `true/false`.

Кроме того, `classList` является перебираемым, поэтому можно перечислить все классы при помощи `for..of`:



Element style

Свойство `elem.style` – это объект, который соответствует тому, что написано в атрибуте `"style"`.

Установка стиля `elem.style.width="100px"` работает так же, как наличие в атрибуте `style` строки `width:100px`.

Для свойства из нескольких слов используется camelCase:

- 1 `background-color` \Rightarrow `elem.style.backgroundColor`
- 2 `z-index` \Rightarrow `elem.style.zIndex`
- 3 `border-left-width` \Rightarrow `elem.style.borderLeftWidth`



Сброс стилей

Иногда нам нужно добавить свойство стиля, а потом, позже, убрать его.

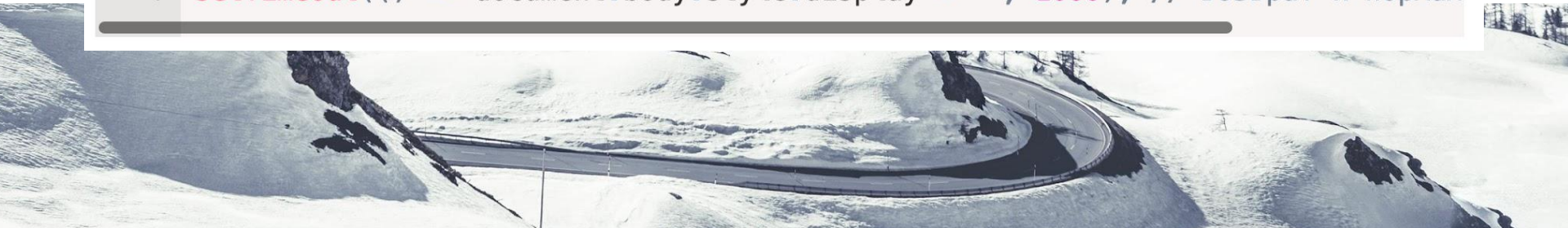
Например, чтобы скрыть элемент, мы можем задать `elem.style.display = "none"`.

Затем мы можем удалить свойство `style.display`, чтобы вернуться к первоначальному состоянию.

Вместо `delete elem.style.display` мы должны присвоить ему пустую строку:

`elem.style.display = ""`.

```
1 // если мы запустим этот код, <body> "мигнёт"
2 document.body.style.display = "none"; // скрыть
3
4 setTimeout(() => document.body.style.display = "", 1000); // возврат к нормал
```



Вычисленные стили: `getComputedStyle`

Итак, изменить стиль очень просто. Но как его *прочитать*?

Например, мы хотим знать размер, отступы, цвет элемента. Как это сделать?

Свойство `style` оперирует только значением атрибута `"style"`, без учёта CSS-каскада.

Поэтому, используя `elem.style`, мы не можем прочесть ничего, что приходит из классов CSS.



Например, здесь `style` не может видеть отступы:

```
1 <head>
2   <style> body { color: red; margin: 5px } </style>
3 </head>
4 <body>
5
6   Красный текст
7   <script>
8     alert(document.body.style.color); // пусто
9     alert(document.body.style.marginTop); // пусто
10  </script>
11 </body>
```



Для этого есть метод: `getComputedStyle`.

Синтаксис:

```
1 getComputedStyle(element, [pseudo])
```

element

Элемент, значения для которого нужно получить

pseudo

Указывается, если нужен стиль псевдоэлемента, например `::before`. Пустая строка или отсутствие аргумента означают сам элемент.

Результат вызова – объект со стилями, похожий на `elem.style`, но с учётом всех CSS-классов.



События





Введение в браузерные события

Событие – это сигнал от браузера о том, что что-то произошло. Все DOM-узлы подают такие сигналы (хотя события бывают и не только в DOM).



События мыши:

- `click` – происходит, когда кликнули на элемент левой кнопкой мыши (на устройствах с сенсорными экранами оно происходит при касании).
- `contextmenu` – происходит, когда кликнули на элемент правой кнопкой мыши.
- `mouseover` / `mouseout` – когда мышь наводится на / покидает элемент.
- `mousedown` / `mouseup` – когда нажали / отжали кнопку мыши на элементе.
- `mousemove` – при движении мыши.



События на элементах управления:

- `submit` – пользователь отправил форму `<form>`.
- `focus` – пользователь фокусируется на элементе, например нажимает на `<input>`.

Клавиатурные события:

- `keydown` и `keyup` – когда пользователь нажимает / отпускает клавишу.

События документа:

- `DOMContentLoaded` – когда HTML загружен и обработан, DOM документа полностью построен и доступен.

CSS events:

- `transitionend` – когда CSS-анимация завершена.





Обработчики событий

Событию можно назначить *обработчик*, то есть функцию, которая сработает, как только событие произошло.

Именно благодаря обработчикам JavaScript-код может реагировать на действия пользователя.

Есть несколько способов назначить событию обработчик. Сейчас мы их рассмотрим, начиная с самого простого.



Использование атрибута HTML

Обработчик может быть назначен прямо в разметке, в атрибуте, который называется `он<событие>`.

Например, чтобы назначить обработчик события `click` на элементе `input`, можно использовать атрибут `onclick`, вот так:

```
1 <input value="Нажми меня" onclick="alert('Клик!')" type="button">
```



При клике мышкой на кнопке выполнится код, указанный в атрибуте `onclick`.



Использование свойства DOM-объекта

Можно назначать обработчик, используя свойство DOM-элемента `on<событие>`.

К примеру, `elem.onclick`:

```
1 <input id="elem" type="button" value="Нажми меня!">
2 <script>
3   elem.onclick = function() {
4     alert('Спасибо');
5   };
6 </script>
```



Так как у элемента DOM может быть только одно свойство с именем `onclick`, то назначить более одного обработчика так нельзя.

В примере ниже назначение через JavaScript перезапишет обработчик из атрибута:

```
1 <input type="button" id="elem" onclick="alert('Было')" value="Нажми меня">  
2 <script>  
3   elem.onclick = function() { // перезапишет существующий обработчик  
4     alert('Станет'); // выведется только это  
5   };  
6 </script>
```



addEventListener

Фундаментальный недостаток описанных выше способов назначения обработчика – невозможность повесить несколько обработчиков на одно событие.

Например, одна часть кода хочет при клике на кнопку делать её подсвеченной, а другая – выдавать сообщение.

Мы хотим назначить два обработчика для этого. Но новое DOM-свойство перезапишет предыдущее:

```
1 input.onclick = function() { alert(1); }  
2 // ...  
3 input.onclick = function() { alert(2); } // заменит предыдущий обработчик
```



Разработчики стандартов достаточно давно это поняли и предложили альтернативный способ назначения обработчиков при помощи специальных методов `addEventListener` и `removeEventListener`. Они свободны от указанного недостатка.

Синтаксис добавления обработчика:

```
1 element.addEventListener(event, handler, [options]);
```

`event`

Имя события, например `"click"`.

`handler`

Ссылка на функцию-обработчик.

`options`

Дополнительный объект со свойствами:





Для удаления обработчика следует использовать `removeEventListener`:

```
1 element.removeEventListener(event, handler[, options]);
```



Объект события

Чтобы хорошо обработать событие, могут понадобиться детали того, что произошло. Не просто «клик» или «нажатие клавиши», а также – какие координаты указателя мыши, какая клавиша нажата и так далее.

Когда происходит событие, браузер создаёт *объект события*, записывает в него детали и передаёт его в качестве аргумента функции-обработчику.



Пример ниже демонстрирует получение координат мыши из объекта события:

```
1 <input type="button" value="Нажми меня" id="elem">
2
3 <script>
4   elem.onclick = function(event) {
5     // вывести тип события, элемент и координаты клика
6     alert(event.type + " на " + event.currentTarget);
7     alert("Координаты: " + event.clientX + ":" + event.clientY);
8   };
9 </script>
```



Полезные ссылки

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

<https://learn.javascript.ru/styles-and-classes>

<https://learn.javascript.ru/introduction-browser-events>





Спасибо!

