



W



**Web Academy**  
Programming Courses

# Front-end с нуля

Тренер: Зинченко Андрей

Лекция 7

[web-academy.com.ua](http://web-academy.com.ua)

A

# Содержание

W

1. setInterval() / setTimeout()
2. Навигация по DOM
3. DOM events
4. JSON



# setTimeout / setInterval



Мы можем вызвать функцию не в данный момент, а позже, через заданный интервал времени. Это называется «планирование вызова».

Для этого существуют два метода:

- `setTimeout` позволяет вызвать функцию **один раз** через определённый интервал времени.
- `setInterval` позволяет вызывать функцию **регулярно**, повторяя вызов через определённый интервал времени.

**W**

# setTimeout / setInterval

A	C
W	
D	M

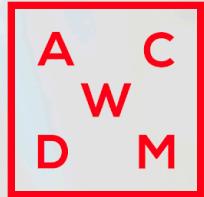
```
1 function sayHi(phrase, who) {  
2   alert( phrase + ', ' + who );  
3 }  
4  
5 setTimeout(sayHi, 1000, "Привет", "Джон") // Привет, Джон
```

```
1 setTimeout(() => alert('Привет'), 1000);
```

**A**

**W**

# setTimeout / setInterval



```
1 let timerId = setTimeout(...);  
2 clearTimeout(timerId);
```

**A**

**W**

# setTimeout / setInterval

A	C
W	
D	M

**A**

```
1 // повторить с интервалом 2 секунды
2 let timerId = setInterval(() => alert('tick'), 2000);
3
4 // остановить вывод через 5 секунд
5 setTimeout(() => { clearInterval(timerId); alert('stop'); }, 5000);
```

**W**

# setTimeout / setInterval



```
1 setTimeout(() => alert("Мир"));
2
3 alert("Привет");
```

**A**

A C  
W M  
D

# W document.getElementById(id)

```
1 <div id="elem">
2   <div id="elem-content">Element</div>
3 </div>
4
5 <script>
6   // получить элемент
7   let elem = document.getElementById('elem');
8
9   // сделать его фон красным
10  elem.style.background = 'red';
11 </script>
```

# W document.getElementById(id)



**⚠ Только `document.getElementById`, а не `anyElem.getElementById`**

Метод `getElementById` можно вызвать только для объекта `document`. Он осуществляет поиск по `id` по всему документу.

**W**

# elem.querySelectorAll(css)



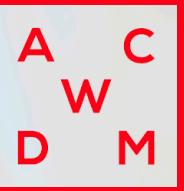
Самый универсальный метод поиска – это `elem.querySelectorAll(css)`, он возвращает все элементы внутри `elem`, удовлетворяющие данному CSS-селектору.

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

**A**

# W

# elem.querySelectorAll(css)



```
1 <ul>
2   <li>Этот</li>
3   <li>тест</li>
4 </ul>
5 <ul>
6   <li>полностью</li>
7   <li>пройден</li>
8 </ul>
9 <script>
10  let elements = document.querySelectorAll('ul > li:last-child');
11
12  for (let elem of elements) {
13    alert(elem.innerHTML); // "тест", "пройден"
14  }
15 </script>
```

# A

# getElementsBy\*



На данный момент, они скорее исторические, так как `querySelector` более чем эффективен.

Здесь мы рассмотрим их для полноты картины, также вы можете встретить их в старом коде.

- `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.

W

# getElementsBy\*

A C  
W  
D M

```
<script>
  let inputs = table.getElementsByTagName('input');

  for (let input of inputs) {
    alert( input.value + ': ' + input.checked );
  }
</script>
```



# Живые коллекции



A

Все методы "getElementsBy\*" возвращают живую коллекцию.  
Такие коллекции всегда отражают текущее состояние документа и  
автоматически обновляются при его изменении.

# Живые коллекции



```
<div>First div</div>
```

```
<script>
  let divs = document.getElementsByTagName('div');
  alert(divs.length); // 1
</script>
```

```
<div>Second div</div>
```

```
<script>
  alert(divs.length); // 2
</script>
```

# Поиск элементов



Метод	Ищет по...	Ищет внутри элемента?	Возвращает живую коллекцию?
querySelector	CSS-selector	✓	-
querySelectorAll	CSS-selector	✓	-
getElementById	id	-	-
getElementsByName	name	-	✓
getElementsByTagName	tag or '*'	✓	✓
getElementsByClassName	class	✓	✓

# Атрибуты элементов



`elem.hasAttribute(name)` – проверяет наличие атрибута.

`elem.getAttribute(name)` – получает значение атрибута.

`elem.setAttribute(name, value)` – устанавливает значение атрибута.

`elem.removeAttribute(name)` – удаляет атрибут.



# Исключения

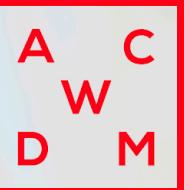
A C  
W M  
D

```
<input id="input" type="checkbox" checked> checkbox

<script>
  alert(input.getAttribute('checked')); // значение атрибута: пустая строка
  alert(input.checked); // значение свойства: true
</script>
```



# Исключения

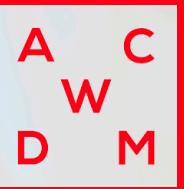


```
<div id="div" style="color:red;font-size:120%">Hello</div>

<script>
    // строка
    alert(div.getAttribute('style')); // color:red;font-size:120%

    // объект
    alert(div.style); // [object CSSStyleDeclaration]
    alert(div.style.color); // red
</script>
```

# Исключения



```
<a id="a" href="#hello">link</a>
<script>
    // атрибут
    alert(a.getAttribute('href')); // #hello

    // свойство
    alert(a.href ); // полный URL в виде http://site.com/page#hello
</script>
```



# Стили и классы



```
let top = /* сложные расчёты */;  
let left = /* сложные расчёты */;
```

```
elem.style.left = left; // например, '123px'  
elem.style.top = top; // например, '456px'
```

A

# className и classList



```
<body class="main page">
  <script>
    alert(document.body.className); // main page
  </script>
</body>
```

A

# className и classList



```
<body class="main page">
  <script>
    // добавление класса
    document.body.classList.add('article');

    alert(document.body.className); // main page article
  </script>
</body>
```

A

# className и classList



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`.

# element.style



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

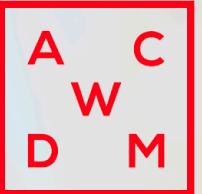
Установка стиля `elem.style.width="100px"` работает так же, как наличие в атрибуте `style` строки `width:100px`.

A

A background image showing a close-up of a laptop keyboard with hands typing. The laptop screen displays some code or text. The overall color palette is cool and professional.

**W**

# element.style



background-color => elem.style.backgroundColor  
z-index => elem.style.zIndex  
border-left-width => elem.style.borderLeftWidth



# element.style

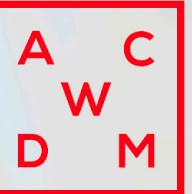


```
<script>
// можем даже устанавливать специальные флаги для стилей, например, "important"
div.style.cssText='color: red !important;
background-color: yellow;
width: 100px;
text-align: center;
';

alert(div.style.cssText);
</script>
```



# element.style



```
<body>
  <script>
    // не работает!
    document.body.style.margin = 20;
    alert(document.body.style.margin); // '' (пустая строка, присваивание игнорируется)

    // сейчас добавим единицу измерения (px) – и заработает
    document.body.style.margin = '20px';
    alert(document.body.style.margin); // 20px

    alert(document.body.style.marginTop); // 20px
    alert(document.body.style.marginLeft); // 20px
  </script>
</body>
```

# getComputedStyle

A	C
W	
D	M

```
<head>
  <style> body { color: red; margin: 5px } </style>
</head>
<body>
```

Красный текст

```
<script>
  alert(document.body.style.color); // пусто
  alert(document.body.style.marginTop); // пусто
</script>
</body>
```

A

# getComputedStyle



```
<head>
  <style> body { color: red; margin: 5px } </style>
</head>
<body>

<script>
  let computedStyle = getComputedStyle(document.body);

  // сейчас мы можем прочитать отступ и цвет

  alert( computedStyle.marginTop ); // 5px
  alert( computedStyle.color ); // rgb(255, 0, 0)
</script>

</body>
```



# Изменение документа

A C  
W M  
D

`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

# Изменение документа

A	C
W	
D	M

```
<script>
  let div = document.createElement('div');
  div.className = "alert";
  div.innerHTML = "<strong>Всем привет!</strong>

  document.body.append(div);
  setTimeout(() => div.remove(), 1000);
</script>
```



# Изменение документа



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.



# Изменение документа



```
<script>
  let div2 = div.cloneNode(true); // клонировать сообщені
  div2.querySelector('strong').innerHTML = 'Всем пока!';
  div.after(div2); // показать клонированный элемент посл
</script>
```

# Изменение документа



- Создать строку, а затем присвоить через `elem.innerHTML`.
- Создавать узлы через методы `DOM`.



# События браузера



*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 браузера



`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 браузера

## 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.

# События браузера



```
<script>
  function countRabbits() {
    for(let i=1; i<=3; i++) {
      alert("Кролик номер " + i);
    }
  }
</script>
```

```
<input type="button" onclick="countRabbits()" value="Считать кроликов!">
```



# События браузера



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





# События браузера

A C  
W M  
D

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

W

# addEventListener

A C  
W M  
D

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

**W**

# addEventListener



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

# addEventListener



`once` : если `true` , тогда обработчик будет автоматически удалён после выполнения.

`capture` : фаза, на которой должен сработать обработчик, подробнее об этом будет рассказано в главе [Всплытие и погружение](#). Так исторически сложилось, что `options` может быть `false/true` , это тоже самое, что `{capture: false/true}` .

`passive` : если `true` , то указывает, что обработчик никогда не вызовет `preventDefault()` , подробнее об этом будет рассказано в главе [Действия браузера по умолчанию](#).

**W**

# addEventListener

A	C
W	
D	M

```
elem.addEventListener( "click" , () => alert('Спасибо!'));  
// ....  
elem.removeEventListener( "click", () => alert('Спасибо!'));
```

```
function handler() {  
    alert( 'Спасибо!' );  
}  
  
input.addEventListener("click", handler);  
// ....  
input.removeEventListener("click", handler);
```

**A**



# addEventListener

A C  
W M  
D

```
<input id="elem" type="button" value="Нажми меня"/>

<script>
    function handler1() {
        alert('Спасибо!');
    }

    function handler2() {
        alert('Спасибо ещё раз!');
    }

    elem.onclick = () => alert("Привет");
    elem.addEventListener("click", handler1); // Спасибо!
    elem.addEventListener("click", handler2); // Спасибо ещё раз!
</script>
```

**W**

# JSON

A	C
W	D
M	

```
{  
    "orders": [  
        {  
            "orderno": "748745375",  
            "date": "June 30, 2088 1:54:23 AM",  
            "trackingno": "TN0039291",  
            "custid": "11045",  
            "customer": [  
                {  
                    "custid": "11045",  
                    "fname": "Sue",  
                    "lname": "Hatfield",  
                    "address": "1409 Silver Street",  
                    "city": "Ashland",  
                    "state": "NE",  
                    "zip": "68003"  
                }  
            ]  
        }  
    ]  
}
```



```
let student = {  
    name: 'John',  
    age: 30,  
    isAdmin: false,  
    courses: ['html', 'css', 'js'],  
    wife: null  
};  
  
let json = JSON.stringify(student);  
  
alert(typeof json); // мы получили строку!  
  
alert(json);  
/* выведет объект в формате JSON:  
{  
    "name": "John",  
    "age": 30,  
    "isAdmin": false,  
    "courses": ["html", "css", "js"],  
    "wife": null  
}  
*/
```



W

# JSON



```
let user = '{ "name": "John", "age": 35, "isAdmin": false, "friends": [0,1,2,3] }'  
  
user = JSON.parse(user);  
  
alert( user.friends[1] ); // 1
```

A