



Веб-разработка

Лекция №5. DOM.

Шумилов Вадим Валерьевич

Тензор, 2016



DOM

DOM – Document Object Model

API для работы со страницей из JavaScript.

- Читать информацию
- Создавать новые элементы
- Манипулировать существующими
 - Изменять свойства
 - Перемещать

DOM – Document Object Model

Все элементы страницы (тэги, текст, комментарии) являются элементами DOM.

Часто его называют DOM-дерево.



Навигация

Каждый DOM-элемент имеет следующие навигационные свойства:

- parentNode – родитель
- previousSibling – предыдущий сосед
- nextSibling – следующий сосед
- firstChild, lastChild – первый и последний дочерний элементы
- childNodes – коллекция дочерних элементов

Важные особенности:

1. Коллекция здесь – не настоящий массив.
2. Если какой-то элемент не определен, значение навигационного свойства будет null.

```
<html>
```

```
<body>
```

```
    <h1>Привет</h1>
```

```
    <p>Текст...</p>
```

```
    <p>Еще абзац</p>
```

```
</body>
```

```
</html>
```



```
<html>
```

```
<body>
```

```
  <h1>Привет</h1>
```

```
  <p>Текст...</p>
```

```
  <p>Еще абзац</p>
```

```
</body>
```

```
</html>
```

```
var body = document.body;
```

```
<html>
```

```
<body>
```

```
  <h1>Привет</h1>
```

```
  <p>Текст...</p>
```

```
  <p>Еще абзац</p>
```

```
</body>
```

```
</html>
```

```
var body = document.body;  
body.firstChild
```

```
<html>  
<body>  
  <h1>Привет</h1>  
  <p>Текст...</p>  
  <p>Еще абзац</p>  
</body>  
</html>
```

```
var body = document.body;  
body.firstChild
```

```
<html>  
<body>  
  <h1>Привет</h1>  
  <p>Текст...</p>  
  <p>Еще абзац</p>  
</body>  
</html>
```

```
var body = document.body;  
body.firstChild  
body.lastChild
```

```
<html>  
<body>  
  <h1>Привет</h1>  
  <p>Текст...</p>  
  <p>Еще абзац</p>  
</body>  
</html>
```

```
var body = document.body;  
var first = body.firstChild;  
first.nextSibling;
```

```
<html>  
<body>  
  <h1>Привет</h1>  
  <p>Текст...</p>  
  <p>Еще абзац</p>  
</body>  
</html>
```

```
var body = document.body;  
var first = body.firstChild;  
first.nextSibling.nextSibling;
```

```
<html>
<body>[
  <h1>Привет</h1>
  <p>Текст...</p>
  <p>Еще абзац</p>
</body>
</html>
```

```
var body = document.body;
var first = body.firstChild;
first.nextSibling.firstChild
```

```
<html>
<body>[
  <h1>Привет</h1>
  <p>Текст...</p>
  <p>Еще абзац</p>
</body>
</html>
```

```
var body = document.body;
var first = body.firstChild;
first.nextSibling.firstChild;
first.nextSibling.firstChild.parentNode
```


Существуют дополнительные навигационные свойства,
не учитывающие текстовые ноды.

- `parentElementNode`
- `firstElementChild`, `lastElementChild`
- `previousElementSibling`, `lastElementSibling`
- `children`

Как работать с коллекциями (childNodes, children)?

Два способа обращения к элементу:

```
elt.childNodes[i];
```

```
elt.childNodes.item(i);
```

Перебор – по индексу от 0 до `elt.childNodes.length`



Поиск элементов

Несколько основных способов найти элемент(ы)

- `getElementById()`
- `getElementsById*()`
- `querySelector()/querySelectorAll()`

```
document.getElementById('elt-id');
```

1. Вызов – с объекта document
2. Возвращает один элемент с указанным id или null

```
anyElement.getElementsBy*('query');
```

1. Вызов – с любого **элемента**. Это ограничивает область поиска
2. Варианты:
 - ByName
 - ByClassName
 - ByTagName
3. Возвращает «**живую** коллекцию» (возможно, пустую)

```
anyElement.querySelector('#css .selector');
```

1. Вызов – с любого **элемента**. Это ограничивает область поиска
2. Возвращает один первый элемент, подходящий под указанный селектор или null, если не найден
3. Бросает исключение, если селектор некорректный

```
anyElement.querySelectorAll('#css .selector');
```

1. Вызов – с любого **элемента**. Это ограничивает область поиска
2. Возвращает коллекцию элементов (возможно, пустую), подходящих под указанный селектор
3. Бросает исключение, если селектор некорректный

Разница между `getElementsBy*` и `querySelectorAll`

Оба метода возвращают коллекцию, но их действие не равнозначно.

`getElementsBy*` возвращают «живую» коллекцию

Поиск элементов



```
<body>  
  <p>Первый</p>  
  <p>Второй</p>  
</body>
```

```
liveCol = document.getElementsByTagName('p');  
col = document.querySelectorAll('p');
```

Поиск элементов



```
liveCol = document.getElementsByTagName('p');  
col = document.querySelectorAll('p');
```

```
liveCol.length; // 2  
col.length; // 2
```

Поиск элементов



```
liveCol = document.getElementsByTagName('p');
```

```
col = document.querySelectorAll('p');
```

```
document.body.innerHTML = ""; // удалит все из body
```

```
liveCol.length; // ??
```

```
col.length; // ??
```

Поиск элементов



```
liveCol = document.getElementsByTagName('p');
```

```
col = document.querySelectorAll('p');
```

```
document.body.innerHTML = ""; // удалит все из body
```

```
liveCol.length; // 0
```

```
col.length; // 2
```



Атрибуты

Для работы с атрибутами у каждого элемента есть следующие методы:

- `getAttribute('name');`
- `setAttribute('name', 'value');`
- `hasAttribute('name');`
- `removeAttribute('name');`

Особенности:

- Значения атрибутов – строки. Все что не строки – конвертируется в строку
- Имена атрибутов - регистронезависимы
- Изменение атрибутов приводит к изменению DOM и HTML

Атрибуты и свойства

- Элементы это объекты.
- Как и у любого объекта, у элементов есть свойства
- Некоторые свойства синхронизируются с атрибутами
 - Некоторые – в обе стороны, некоторые – только в одну

```
<div id="some"></div>
```

```
var elt = document.getElementById('some');
```

```
elt.id; // 'some'
```

```
elt.getAttribute('id'); // 'some'
```

```
<div id="some"></div>
```

```
var elt = document.getElementById('some');
```

```
elt.id = "foo";
```

```
elt.getAttribute('id'); // 'foo'
```

```
elt.outerHTML; // '<div id="foo"></div>'
```

Не все атрибуты и свойства синхронизируются

```
<input type="text" value="foo" />
```

```
var elt = document.getElementsByTagName('input')[0];
```

```
elt.value; // 'foo'
```

```
elt.getAttribute('value'); // 'foo'
```

Не все атрибуты и свойства синхронизируются

```
<input type="text" value="foo" />
```

```
var elt = document.getElementsByTagName('input')[0];
```

```
elt.value = 'bar';
```

```
elt.getAttribute('value'); // 'foo'
```

```
elt.innerHTML; // '<input type="text" value="foo" />'
```

Но на экране в поле ввода будет "bar"...



Модификация дерева

Модификация дерева



Можно создавать ноды дерева

```
var div = document.createElement('div');
```

```
var text = document.createTextNode('Это текстовая нода');
```

Модификация дерева



Можно добавлять ноды в другие ноды

```
div.appendChild(text);
```

```
document.body.appendChild(div);
```


Модификация дерева



Можно читать что получилось в виде строки:

```
document.body.innerHTML;
```

```
<div>Это текстовая нода</div>
```

Модификация дерева



Можно читать что получилось в виде строки:

```
document.body.outerHTML;
```

```
<body><div>Это текстовая нода</div></body>
```

Модификация дерева



Создавать содержимое документа можно с помощью `innerHTML`.

```
document.body.innerHTML = "<p>Параграф текста</p>";
```

«Перезапись» `innerHTML` приводит к удалению всего содержимого, которое ранее было внутри.

Модификация дерева



Можно управлять местом вставки

```
parent.insertBefore(elem, nextSibling);  
parent.replaceChild(newElem, elem);
```

Модификация дерева



insertBefore

```
<body>  
    <div id="ref"></div>  
</body>
```

Модификация дерева



insertBefore

```
<body>
```

```
  <div id="ref"></div>
```

```
</body>
```

```
var ref = document.getElementById('ref');  
var newDiv = document.createElement('div');  
document.body.insertBefore(newDiv, ref);
```

Модификация дерева



insertBefore

```
var ref = document.getElementById('ref');  
var newDiv = document.createElement('div');  
document.body.insertBefore(newDiv, ref);
```

```
<body>  
  <div></div>  
  <div id="ref"></div>  
</body>
```

Модификация дерева



Если указать последний параметр `null` – вставка будет эквивалентна `appendChild`.

```
insertBefore(newChild, null) == appendChild(newChild)
```


Модификация дерева



replaceChild

```
<body>
```

```
    <div id="ref"></div>
```

```
</body>
```

```
var ref = document.getElementById('ref');  
var newDiv = document.createElement('div');  
document.body.replaceChild(newDiv, ref);
```

Модификация дерева



replaceChild

```
var ref = document.getElementById('ref');  
var newP = document.createElement('p');  
document.body.replaceChild(newDiv, ref);
```

```
<body>  
    <p></p>  
</body>
```

Модификация дерева



Элементы можно удалять методом `removeChild`

```
<body>  
  <div id="ref"></div>  
</body>
```

```
var ref = document.getElementById('ref');  
document.body.removeChild(ref);
```

Модификация дерева



Элементы можно удалять методом `removeChild`

```
var ref = document.getElementById('ref');  
document.body.removeChild(ref);
```

```
<body>
```

```
</body>
```

Модификация дерева



Элементы можно клонировать методом cloneNode

```
<body>
```

```
  <p>Абзац</p>
```

```
</body>
```

```
var p = document.body.firstChild;
```

```
var pClone = p.cloneNode();
```

```
document.body.appendChild(pClone);
```

Модификация дерева



Элементы можно клонировать методом cloneNode

```
var p = document.body.firstChild;  
var pClone = p.cloneNode();  
document.body.appendChild(pClone);
```

```
<body>  
  <p>Абзац</p>  
  ???  
</body>
```

Модификация дерева



Элементы можно клонировать методом `cloneNode`

```
var p = document.body.firstChild;  
var pClone = p.cloneNode();  
document.body.appendChild(pClone);
```

```
<body>  
  <p>Абзац</p>  
  <p></p>  
</body>
```

Модификация дерева



Элементы можно клонировать методом cloneNode

```
var p = document.body.firstChild;  
var pClone = p.cloneNode(true);  
document.body.appendChild(pClone);
```

```
<body>  
  <p>Абзац</p>  
  <p>Абзац</p>  
</body>
```




События

DOM Events. События. Какие они бывают?

DOM Events. События. Какие они бывают?

- Click
- Mousedown, Mousemove
- Focus
- Keydown, Keyup
- Submit

Как добавить обработчик события.

```
<p onclick="alert('Нажали на текст')">...</p>
```

```
p.onclick=function() {  
    alert('Нажали на текст');  
};
```

Как добавить обработчик события.

```
function handleClick() { ... };
```

```
<p onclick="handleClick()">...</p>
```

```
p.onclick=handleClick;
```

```
<p onclick="handleClick()">...</p>
```

```
p.onclick=handleClick;
```

```
p.setAttribute('onclick', handleClick);
```

```
<p onclick="handleClick()">...</p>
```

```
p.onclick=handleClick;
```

```
p.setAttribute('onclick', handleClick);
```

```
p.onclick = f1;
```

```
p.onclick = f2;
```



```
p.onclick = f1;
```

```
p.onclick = f2;
```

Проблема. Второй обработчик «затрет» первый!

addEventListener!

```
p.addEventListener('click', f1);
```

```
p.addEventListener('click', f2);
```

Обратите внимание!

```
p.onclick = ...
```

```
p.addEventListener('click', ...);
```

При навешивании через атрибут нужно добавить on!

`elt.addEventListener` vs `elt.on*`

1. Позволяет добавить несколько обработчиков
2. Есть события, которые нельзя добавить через свойство/разметку. Можно только через `addEventListener`

Как удалить обработчик?

```
elt.onclick = null;
```

```
elt.removeEventListener('click', f);
```



```
elt.addEventListener('click', function() { doSmth(); });
```

```
elt.removeEventListener('click', function() { doSmth();});
```



```
elt.addEventListener('click', function() { doSmth(); });
```

```
elt.removeEventListener('click', function() { doSmth();});
```

Важно! Требуется указывать всегда одну и ту же функцию!

```
function handler() {  
    doSmth();  
}
```

```
elt.addEventListener('click', handler);  
elt.removeEventListener('click', handler);
```




Объект события

Объект события



```
p.onclick = function(event) {  
    // event???  
}
```

Объект события



Объект-дескриптор события. Содержит свойства и методы, позволяющие получить дополнительную информацию о событии, управлять им и т.п.

Объект события



`event.type` – тип события (click, mousedown, keyup, etc.)

`event.target` – объект, на котором случилось событие

Есть разные специфические свойства

`event.clientX`, `event.clientY` – координаты курсора в момент клика

Объект события



Как получить доступ к событию, если обработчик навешивается через атрибут?

Объект события



Как получить доступ к событию, если обработчик навешивается через атрибут?

```
<div onclick="alert(event.clientX)">...</div>
```

Объект события



Как получить доступ к событию, если обработчик навешивается через атрибут?

```
<div onclick="alert(event.clientX)">...</div>
```

Это эквивалентно ...

```
div.onclick = function(event) {  
    alert(event.clientX);  
}
```



Всплытие события


```
<div onclick="alert('Click!')">
```

```
  <p>Первый
```

```
    <span>параграф</span>
```

```
  </p>
```

```
</div>
```

```
<div onclick="alert('Click!')">
```

```
  <p>Первый
```

```
    <span>параграф</span>
```

```
  </p>
```

```
</div>
```

```
<div onclick="alert('Click!')">
```

```
  <p>Первый
```

```
    <span>параграф</span>
```

```
  </p>
```

```
</div>
```

```
<div onclick="alert('Click!')">
```

```
  <p>Первый
```

```
    <span>параграф</span>
```

```
  </p>
```

```
</div>
```

`event.target` – элемент, на котором изначально случилось событие

`event.currentTarget` – элемент, на котором событие было поймано

Всплытие можно заблокировать

```
<div onclick="alert('Click!')">  
  <p onclick="event.stopPropagation()">  
    <span>....</span>  
  </p>  
</div>
```

Всплытие можно заблокировать

```
<div onclick="alert('Click!')">  
  <p onclick="event.stopPropagation()">  
    <span>....</span>  
  </p>  
</div>
```

В этом примере мы не увидим alert();

Что если обработчиков несколько?

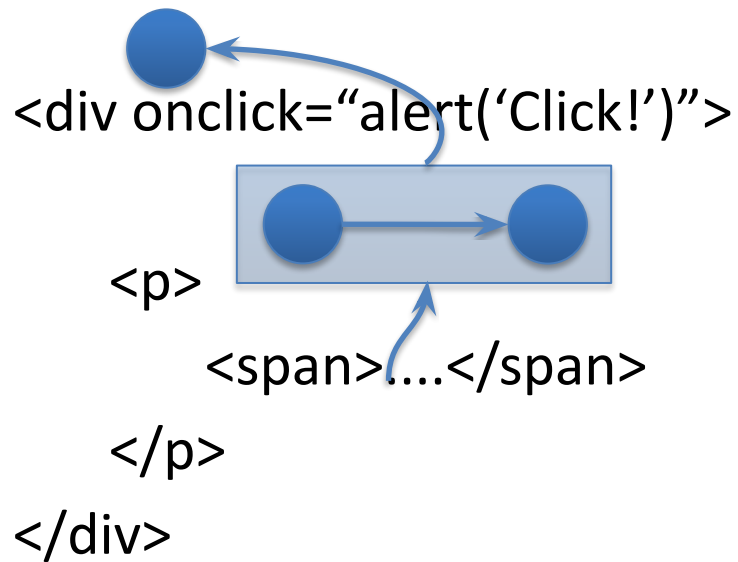
```
<div onclick="alert('Click!')">  
    <p><span>....</span></p>  
</div>
```

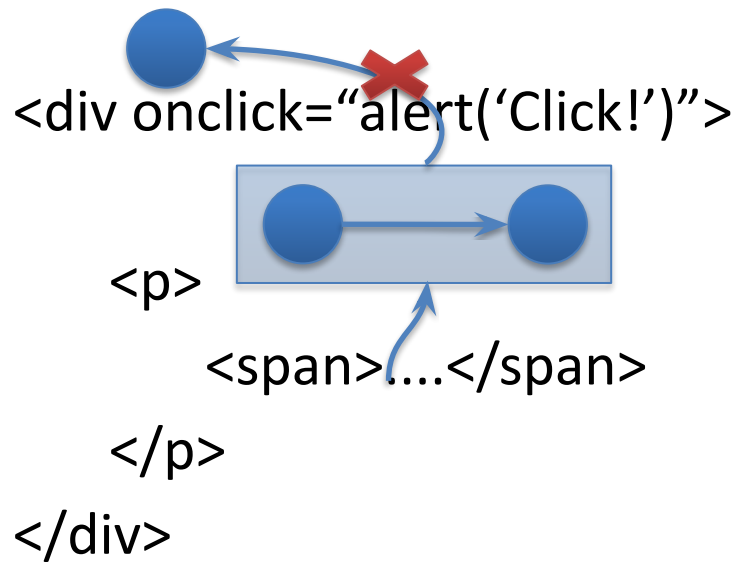
```
var p = document.getElementsByTagName('p')[0];  
p.addEventListener('click', stopsPropagation);  
p.addEventListener('click', showsAlert);
```



```
function stopsPropagation(event) {  
    event.stopPropagation();  
}
```

```
function showAlert(event) {  
    alert(event.target.tagName); // SPAN  
}
```







`<div onclick="alert('Click!')">`



`<p>`

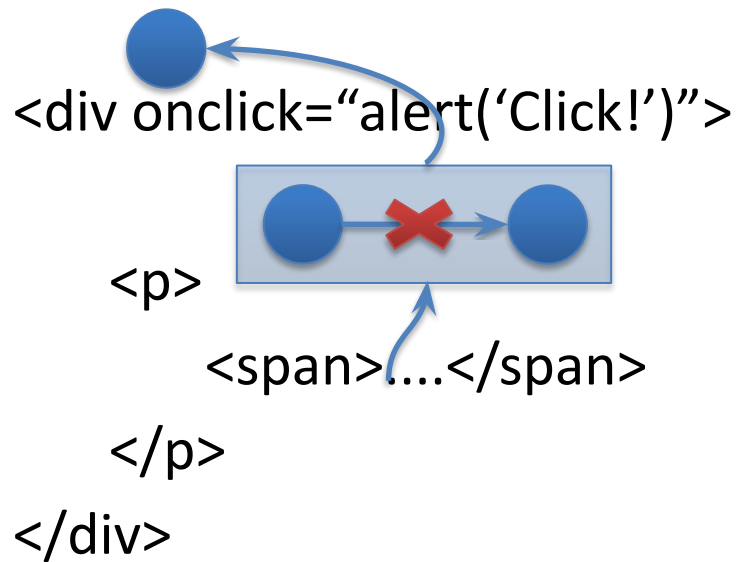
`<span>...</span>`

`</p>`

`</div>`

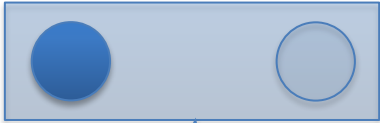
```
function stopsPropagation(event) {  
    event.stopImmediatePropagation();  
}
```

```
function showAlert(event) {  
    alert(event.target.tagName); // SPAN  
}
```





`<div onclick="alert('Click!')">`



`<p>`

`<span>...</span>`

`</p>`

`</div>`

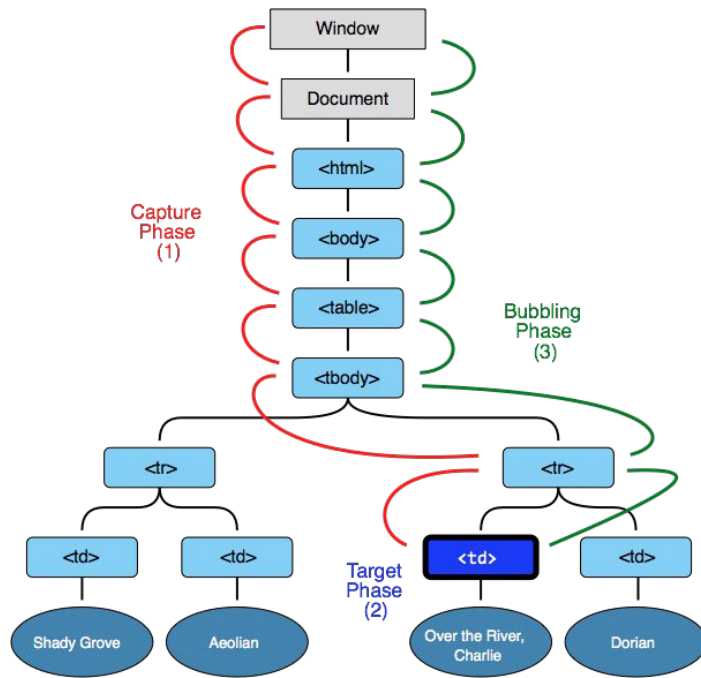
Всплытие



Не все события всплывают!

Пример такого события – focus.

На самом деле есть не только всплытие, но и погружение событий!



Фазой, на которой будет обрабатываться событие, управляет третий, необязательный аргумент `addEventListener`

```
elt.addEventListener('click', handler, true);
```

Для того, чтобы поймать событие на стадии «погружения», нужно передать его `== true`

Всплытие



Кстати, событие focus...

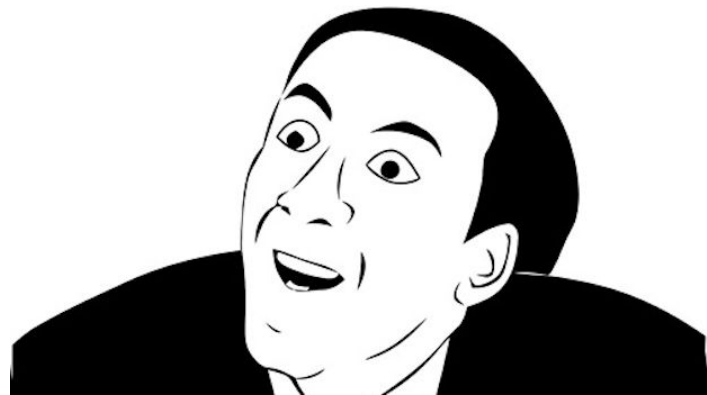
Оно не всплывает, но погружается.

Всплытие



Кстати, событие focus...

Оно не всплывает, но погружается.





Делегирование

Делегирование



Задача: хотим при нажатии на каждый абзац в документе считать сколько в нем слов.

Решение 1 (плохое)

```
var pCol = document.getElementsByTagName('p');  
for(var i = 0, l = pCol.length; i < l; i++) {  
    pCol[i].addEventListener('click', countWords);  
}
```

Делегирование



Чем плохо это решение?

Чем плохо это решение?

1. Если элементов будет много их перебор может занять время
2. Навешивается обработчиков по количеству параграфов
3. Если в документ добавятся новые параграфы для них код работать не будет, ведь они появились позже!

Делегирование



Решение 2 (почти правильное). Делегирование!

```
document.body.addEventListener('click', function(event) {  
    if (event.target.nodeName == 'P') {  
        countWords(event.target);  
    }  
});
```

Решение 2 (почти правильное). Делегирование!

- ~~1. Если элементов будет много их перебор может занять время~~
2. Навешивается обработчиков по количеству параграфов
3. Если в документ добавятся новые параграфы для них код работать не будет, ведь они появились позже!

Решение 2 (почти правильное). Делегирование!

- ~~1. Если элементов будет много их перебор может занять время~~
- ~~2. Навешивается обработчиков по количеству параграфов~~
3. Если в документ добавятся новые параграфы для них код работать не будет, ведь они появились позже!

Решение 2 (почти правильное). Делегирование!

- ~~1. Если элементов будет много их перебор может занять время~~
- ~~2. Навешивается обработчиков по количеству параграфов~~
- ~~3. Если в документ добавятся новые параграфы для них код работать не будет, ведь они появились позже!~~

Делегирование



Решение 2 (почти правильное). Делегирование!

Делегирование



Решение 2 (почти правильное). Делегирование!

<p>

Текст, часть которого

выделена жирным

</p>

Решение 3 (правильное)

```
function (event) {  
    var target = event.target;  
    while (target) {  
        if (target.nodeName == 'P') break;  
        target = target.parentNode  
    }  
    if (target) countWords(target);  
}
```




Действие по умолчанию

Действие по умолчанию



У многих событий есть действия по умолчанию:

- Click по ссылке – переход
- Mousedown на поле ввода – фокусировка
- Keydown – при нажатии клавиши в поле ввода там появляется символ

Действие по умолчанию



Действие по умолчанию можно отменить

```
elt.addEventListener('click', function(event) {  
    event.preventDefault();  
});
```

Действие по умолчанию



Действие по умолчанию можно отменить

Если событие навешено через атрибут, можно короче

```
elt.onclick = function() {  
    return false;  
};
```

ВНИМАНИЕ! Это не работает, если обработчик навешен через `addEventListener`!

Действие по умолчанию



Для чего это может применяться?

- Для отмены перехода по ссылке и выполнения вместо этого какого-либо другого действия.
- Для отмены ввода в `<input ...>`, например, с целью коррекции ввода



Полезные ссылки

- <http://javascript.ru/>
- <http://learn.javascript.ru/>
- <http://learn.javascript.ru/document>
- <http://learn.javascript.ru/events-and-interfaces>



Вопросы есть?



Спасибо за внимание!