**Обращение к свойствам через свойства document**

Некоторые теги страницы можно не получать методами типа **getElementById**, а обратиться к ним как свойствам объекта **document**. К примеру, к тегу **<body>** можно обратиться так: **document.body**, а к тегу **<head>** - вот так: **document.head**. Такое работает не для всех тегов, а только для избранных.

Учтите, что нельзя получить доступ к тому свойству, которого еще не существует в момент выполнения скрипта. Поэтому, если скрипт находится в **<head>**, то в нем недоступен **document.body**.

В следующем примере первый alert выведет **null**:

<!DOCTYPE HTML>

<html>

<head>

<script>

alert(document.body); //null, так как body еще нет

</script>

</head>

<body>

<script>

alert(document.body); //body уже есть

</script>

</body>

</html>

**Работа с формами**

Можно получить все HTML формы на странице через **document.forms**. В результате мы получим массив форм, будто он был получен с помощью getElementsByTagName (точнее это будет не массив, а [псевдомассив (или коллекция)](http://theory.phphtml.net/javascript/pages/pseudo-array.html) свойств).

Мы можем, к примеру, обратиться к любой форме, как к энному свойству массива, например, к форме с номером 0:

<body>

<form>

<input value="1">

<input value="2">

<input value="3">

</form>

<form>

<input value="1">

<input value="2">

<input value="3">

</form>

</body>

document.forms[0].innerHTML **=** '!';

HTML код станет выглядеть так:

<body>

<form>

!

</form>

<form>

<input type="text" value="1">

<input type="text" value="2">

<input type="text" value="2">

</form>

</body>

А можем перебрать все формы с помощью цикла:

<body>

<form>

<input value="1">

<input value="2">

<input value="3">

</form>

<form>

<input value="1">

<input value="2">

<input value="3">

</form>

</body>

var forms **=** document.forms;

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

forms[i].innerHTML **=** '!';

}

HTML код станет выглядеть так:

<body>

<form>

!

</form>

<form>

!

</form>

</body>

Следующем образом - **document.forms[i].elements** - можно найти общее количество форм на странице.

Кроме того, к формам можно обращаться по имени вместо номера. Это имя задается в атрибуте **name** тега **<form>**. Например, если форме дать имя test, то к ней можно обратиться так - **document.forms['test']** или так - **document.forms.test**.

В следующем примере форме с именем **form1** поставим текст '!', а форме с именем **form2** - текст '?' (двумя разными способами):

<body>

<form name="form1">

<input value="1">

<input value="2">

<input value="3">

</form>

<form name="form2">

<input value="1">

<input value="2">

<input value="3">

</form>

</body>

document.forms.form1.innerHTML **=** '!';

document.forms['form2'].innerHTML **=** '?';

HTML код станет выглядеть так:

<body>

<form name="form1">

!

</form>

<form name="form2">

?

</form>

</body>

**Обращение к свойствам форм**

У каждой формы есть свойство **elements**, которое содержит массив свойств формы. С его помощью можно обратиться к определенному свойству формы по его номеру.

Для примера давайте обратимся к форме номер 1, а затем к ее свойству номер 0:

<body>

<form>

<input value="1">

<input value="2">

<input value="3">

</form>

<form>

<input value="1">

<input value="2">

<input value="3">

</form>

</body>

document.forms[1].elements[0].value **=** '!';

HTML код станет выглядеть так:

<body>

<form>

<input value="1">

<input value="2">

<input value="3">

</form>

<form>

<input value="!">

<input value="2">

<input value="3">

</form>

</body>

Можно также перебрать все свойства формы с помощью цикла. Давайте получим форму с номером 1 и всем свойствам этой формы поставим **value** '!':

<body>

<form>

<input value="1">

<input value="2">

<input value="3">

</form>

<form>

<input value="1">

<input value="2">

<input value="3">

</form>

</body>

var elements **=** document.forms[1].elements;

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

elements[i] **=** '!';

}

HTML код станет выглядеть так:

<body>

<form>

<input value="1">

<input value="2">

<input value="3">

</form>

<form>

<input value="!">

<input value="!">

<input value="!">

</form>

</body>

При необходимости с помощью двух вложенных циклов можно перебрать все формы и все свойства внутри них.

К свойствам формы, так же, как и к самим формам, можно обращаться по имени из атрибута **name**:

<body>

<form name="form1">

<input value="1" name="input1">

<input value="2" name="input2">

<input value="3" name="input3">

</form>

</body>

document.forms.form1.elements.input1.value **=** '!';

HTML код станет выглядеть так:

<body>

<form name="form1">

<input value="!" name="input1">

<input value="2" name="input2">

<input value="3" name="input3">

</form>

</body>

Это не все приемы работы с формами, при необходимости смотрите подробности в справочнике JavaScript тут: работа с формами.

# Работа с событиями

**Работа с addEventListener**

Метод **addEventListener** первым параметром принимает название события, а вторым - функцию, которую нужно привязать к этому событию. При этом имя события пишется без 'on': 'click' вместо 'onclick', 'mouseover' вместо 'onmouseover' и так далее. Имя функции (второй параметр) пишется без кавычек и без круглых скобок (зачем это нужно, мы с вами уже разобрали выше).

Давайте сделаем так, чтобы по клику на кнопку вызывалась функция func:

<input type="submit" id="test">

var elem **=** document.getElementById('test');

elem.addEventListener('click', func);

function func() {

alert('!');

}

Привяжем теперь одновременно два события. При этом события не будут затирать друг друга, сколько бы их не объявили:

<input type="submit" id="test">

var elem **=** document.getElementById('test');

elem.addEventListener('click', func1);

elem.addEventListener('click', func2);

function func1() {

alert('1');

}

function func2() {

alert('2');

}

Если вы скопируете этот код и запустите его у себя - сработает и функция func1, и функция func2.

**Работа с this для addEventListener**

Мы с вами уже разбирали работу с **this** в случае, если привязывать события вторым способом (через свойство элемента). В случае с addEventListener **this** работает аналогичным образом - он будет ссылаться на тот элемент, в котором случилось событие.

Давайте посмотрим на примере: привяжем к кнопке событие **onclick**, которое будет запускать функцию **func**. Эта функция будет выводить на экран **value** нашей кнопки:

<input type="submit" id="test" value="!">

var elem **=** document.getElementById('test');

elem.addEventListener('click', func);

function func() {

alert(this.value);

}

С одной кнопкой не очень интересно проверять работу **this**. Давайте сделаем две кнопки, привязав в ним одну и ту же функцию **func**. В этом случае функция func будет выводить value той кнопки, на которую мы кликнули:

<input type="submit" id="test1" value="Кнопка 1">

<input type="submit" id="test2" value="Кнопка 2">

var elem1 **=** document.getElementById('test1');

elem1.addEventListener('click', func);

var elem2 **=** document.getElementById('test2');

elem2.addEventListener('click', func);

function func() {

alert(this.value);

}

Здесь удобство работы с **this** в том, что не нужно создавать разные функции для разных элементов. Есть одна функция **func**, которая делает одно и то же, но для разных элементов и различаем мы их через **this** - на какой элемент кликнули - тот элемент и будет в **this**.

Ну, а сейчас получим массив кнопок с помощью getElementsByTagName и каждой из них привяжем функцию func.

В **this** будет лежать ссылка на ту кнопку, на которую вы нажали, и функция **func** выведет на экран именно ее **value**:

<input type="submit" value="Кнопка 1">

<input type="submit" value="Кнопка 2">

<input type="submit" value="Кнопка 3">

var elems **=** document.getElementsByTagName('input');

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

elems[i].addEventListener('click', func);

}

function func() {

alert(this.value);

}

**Удаление привязки через removeEventListener**

Сейчас мы с вами займемся тем, что будем удалять привязанные к событию функции. Что у нас будет получаться: если, к примеру, к событию onclick привязаны функции **func1** и **func2**, то мы сможем отвязать функцию func1, не затрагивая func2 и наоборот.

Давайте привяжем к элементу 3 функции: func1, func2 и func3, которые будут выводить на экран числа 1, 2 и 3:

<input type="submit" id="test">

var elem **=** document.getElementById('test');

elem.addEventListener('click', func1);

elem.addEventListener('click', func2);

elem.addEventListener('click', func3);

function func1() {

alert('1');

}

function func2() {

alert('2');

}

function func3() {

alert('3');

}

А теперь сразу же после привязки отвяжем функции func1 и func2. Это делается с помощью метода **removeEventListener**, которая принимает те же параметры, что и addEventListener:

<input type="submit" id="test">

var elem **=** document.getElementById('test');

elem.addEventListener('click', func1);

elem.addEventListener('click', func2);

elem.addEventListener('click', func3);

elem.removeEventListener('click', func1);

elem.removeEventListener('click', func2);

function func1() {

alert('1');

}

function func2() {

alert('2');

}

function func3() {

alert('3');

}

Если вы запустите этот пример, то увидите, что сработает функция func3, а первые две - нет.

Понятно, что по сути мы сделали бессмысленную операцию: привязали функции и сразу же отвязали. Давайте усложним пример и будем удалять привязку внутри самих функций.

Пусть при первом клике на кнопку сработают все 3 функции и при этом func1 и func2 отвяжутся от элемента. И при следующих кликах будет срабатывать только функция func3, которую мы не отвязываем.

<input type="submit" id="test">

var elem **=** document.getElementById('test');

elem.addEventListener('click', func1);

elem.addEventListener('click', func2);

elem.addEventListener('click', func3);

function func1() {

alert('1');

//Отвяжем функцию func1:

this.removeEventListener('click', func1);

}

function func2() {

alert('2');

//Отвяжем функцию func2:

this.removeEventListener('click', func2);

}

function func3() {

alert('3');

}

Обратите внимание на **this** внутри функции - он указывает на наш элемент.

А в следующем примере мы ко всем кнопкам привяжем функцию func, которая будет выводить содержимое атрибута value той кнопки, на которую вы нажмете. А после этого функция func будет отвязываться от этой кнопки с помощью **removeEventListener**. И получится что каждая кнопка будет реагировать только на первое нажатие по ней (запустите код и проверьте это):

<input type="submit" value="Кнопка 1">

<input type="submit" value="Кнопка 2">

<input type="submit" value="Кнопка 3">

var elems **=** document.getElementsByTagName('input');

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

elems[i].addEventListener('click', func);

}

function func() {

alert(this.value);

this.removeEventListener('click', func);

}

Задания

1. Даны ссылки. Привяжите всем ссылкам событие - по наведению на ссылку в атрибут title запишется ее текст.
2. Привяжите всем ссылкам событие - по наведению на ссылку в конец ее текста дописывается ее href в круглых скобках.
3. Дополните предыдущую задачу: после первого наведению на ссылку следует отвязать от нее событие, которое добавляет href в конец текста.
4. Привяжите всем инпутам следующее событие - по потери фокуса каждый инпут выводит свое value в абзац с id="test".
5. Для всех инпутов сделайте так, чтобы они выводили свой value алертом при нажатии на любой из них, но только по первому нажатию. Повторное нажатие на инпут не должно вызывать реакции.
6. Даны абзацы с числами. По нажатию на абзац в нем должен появится квадрат числа, которое он содержит.
7. Даны инпуты. Сделайте так, чтобы все инпуты по потери фокуса проверяли свое содержимое на правильное количество символов. Сколько символов должно быть в инпуте, указывается в атрибуте data-length. Если вбито правильное количество, то граница инпута становится зеленой, если неправильное - красной.
8. Даны дивы. По первому нажатию на каждый див он красится красным фоном, по второму красится обратно и так далее каждый клик происходит чередование фона. Сделайте так, чтобы было две функции: одна красит в красный цвет, другая в зеленый и они сменяли друг друга через removeEventListener.
9. Привяжите всем ссылкам событие - по наведению на ссылку в конец ее текста дописывается ее href в круглых скобках.
10. Дополните предыдущую задачу: после первого наведению на ссылку следует отвязать от нее событие, которое добавляет href в конец текста.
11. Привяжите всем инпутам следующее событие - по потери фокуса каждый инпут выводит свое value в абзац с id="test".
12. Для всех инпутов сделайте так, чтобы они выводили свой value алертом при нажатии на любой из них, но только по первому нажатию. Повторное нажатие на инпут не должно вызывать реакции.