**Продвинутая работа с событиями в JavaScript**

*В данном уроке мы с вами разберем все способы работы с событиями в JavaScript.*

**События через атрибуты**

Вы уже хорошо знаете **первый** способ привязать событие к элементу - с помощью атрибута, например **onclick** (если вы не помните этого или пропустили - см. урок [основы работы с событиями JavaScript](http://code.mu/books/javascript/dom/osnovy-raboty-s-dom-v-javascript.html)).

Напомню еще раз этот способ на примере: сейчас по клику на кнопку сработает функция **func**, которая выводит на экран **alert**:

<input type="submit" onclick="func()">

function func() {

alert('!');

}

Задания события **через атрибут** имеет свои недостатки: если, к примеру, я хочу ко **всем**инпутам на странице привязать определенное событие - мне придется полазить по HTML странице, найти все инпуты и каждому добавить атрибут с событием. Это не очень удобно и хотелось бы иметь под рукой более продвинутый способ работы с событиями.

Поэтому давайте посмотрим, что еще нам может предложить **JavaScript** при работе с событиями.

**События через работу с атрибутами**

По сути атрибут **onclick** является таким же атрибутом, как, к примеру, **value**. И, если мы могли менять атрибут **value** таким образом - *elem.value*, то точно также мы можем менять атрибут **onclick**.

Если мы сделаем так: *elem.onclick = func*, то привяжем к элементу **elem** функцию **func**. Посмотрите пример и под ним мы обсудим все **нюансы** этого способа:

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

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

elem.onclick **=** func;

function func() {

alert('!');

}

Теперь я должен открыть вам страшную тайну JavaScript: если функция написана без круглых скобок, например *func*, то она возвращает свой **исходный код**, а если с круглыми скобками, например *func()*, то возвращает **результат** работы функции.

Я уверен, что прочитав это, вы не до конца поймете то, что я хотел вам донести, поэтому запустите пример и еще раз перечитайте предыдущий абзац. Вот пример:

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

//Функция возвращает строку '!'

function func() {

return '!';

}

alert(func()); //вы увидите строку '!'

alert(func); //а сейчас вы увидите исходный код функции func!

Теперь, зная эту страшную тайну, вернемся к строке *elem.onclick = func* - в данном случае я в атрибут **onclick** записываю **исходный код** функции, а не ее результат - и все работает. Если вы сделаете так - *elem.onclick = func()* - то запишите результат функции и ничего не будет работать.

Кстати, результатом функции **func()** будет *undefined*, так как у нее нет команды **return**. Напомню код функции, о которой идет речь:

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

function func() {

alert('!');

}

Давайте вспомним метод **setInterval** (см. урок [работа с таймерами в JavaScript](http://code.mu/books/javascript/dom/rabota-s-tajmerami-v-javascript.html)), когда мы использовали его таким образом *window.setInterval(timer, 1000)* - в этом случае мы также писали функцию **timer** без круглых скобок, потому что нас интересовал не результат работы функции, а ее **код**.

Мы еще поговорим подробнее о все этих нюансах, пока запомните эти вещи, которые я рассказал, пусть даже без полного понимания. Оно со временем придет.

**Достоинства и недостатки такого способа**

Давайте теперь обсудим достоинства и недостатки этого способа.

**Достоинства**

**Достоинства** такие: теперь, если мы захотим задать одновременно одно событие всем инпутам на странице, нет нужды вручную прописывать каждому из них атрибут с событием. Можно просто получить все инпуты, перебрать их циклом и каждому привязать событие.

Давайте сделаем это. Получим все инпуты с помощью [getElementsByTagName](http://code.mu/javascript/dom/getElementsByTagName.html) и в цикле привяжем каждому такое событие: пусть по клику каждый инпут выдает алертом текст '!':

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

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

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

<input type="submit" value="4">

<input type="submit" value="5">

/\*

Этот код будет запускаться по загрузке страницы.

Напоминаю, что писать его нужно после HTML кода.

\*/

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

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

elems[i].onclick **=** func;

}

function func() {

alert('!');

}

Теперь нажатие на любой инпут будет приводить к тому, что будет срабатывать функция **func**, которая алертом выводит '!'.

**Использование this**

Давайте усложним задачу и сделаем так, чтобы **alert** выводил содержимое атрибута **value** того инпута, на который кликнули мышкой.

Для этого нужно воспользоваться **this**, только не так, как мы это делали раньше. Раньше, когда мы писали событие прямо в атрибут, мы делали так: *onclick="func(this)"*, однако сейчас вот так - *elems[i].onclick = func(this)* - мы сделать **не можем**.

**Во-первых**, потому, что функцию **func** здесь нужно писать без круглых скобок, иначе будет не исходный код, а **результат** работы функции.

**Во-вторых**, this указывает на разные элементы в зависимости от *контекста* (в зависимости от места, где он написан). И в нашем случае он не будет ссылаться на нужный нам элемент (то есть на тот, на который мы кликнули). Про контекст выполнения поговорим чуть позже, пока просто знайте, что он есть.

Вы можете спросить, почему тут - *onclick="func()"* - в функции написаны круглые скобки, хотя по логике там тоже требуется исходный код, а не результат. Об этом вы узнаете в уроке про анонимные функции чуть позже.

Так как правильно использовать this в нашей конструкции *elems[i].onclick = func*? На самом деле тут **this** доступен внутри функции **func** и он ссылается на тот элемент, в котором возникло событие, вызвавшее эту функцию. То есть, если я делаю клик по **первому** инпуту - в **this** будет лежать ссылка на него, а если по **второму** - то на него.

В данном случае считайте, что **this** - это будто переменная **elem**, полученная через **getElementById**. К примеру, *elem.value* позволяет обратиться к атрибуту **value**, значит *this.value* будет делать то же самое.

Итак, давайте все-таки решим нашу задачу - сделаем так, чтобы **alert** выводил содержимое атрибута **value** того инпута, на который **кликнули** мышкой:

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

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

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

<input type="submit" value="4">

<input type="submit" value="5">

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

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

elems[i].onclick **=** func;

}

function func() {

alert(this.value); //изменение только здесь

}

Учтите, что если вы попытаетесь воспользоваться **this** внутри функции, которая была прописана прямо в атрибуте (то есть первым способом задать событие) - у вас ничего не выйдет. Смотрите пример **неправильного** кода:

<input type="submit" value="Кнопка!" onclick="func()">

function func() {

alert(this.value); //не выведет ожидаемого

}

В данном случае контекст выполнения таков, что **this** ссылается на **window**, а не на текущий элемент. Почему так - поговорим, когда вы разберете анонимные функции.

Напоминаю правильный вариант:

<input type="submit" value="Кнопка!" onclick="func(this)">

function func(elem) {

alert(elem.value); //выведет содержимое атрибута value

}

**Недостатки**

Теперь поговорим о **недостатках**. Недостатком такого способа будет то, что мы можем привязать к событию только одну функцию. Если попытаться сначала записать одну функцию, а потом другую - у нас ничего не получится.

В следующем примере мы пытаемся привязать к событию **onclick** сразу две функции **func1** и **func2**. Однако по клику на элемент сработает только вторая функция, так как она затрет первую:

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

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

elem.onclick **=** func1; //тут мы сначала привязали func1

elem.onclick **=** func2; //а теперь func2 и func1 уже не привязана

function func1() {

alert('1');

}

function func2() {

alert('2');

}

В принципе, эту проблему легко обойти, если ввести еще и третью функцию func3. Привяжем к атрибуту onclick только func3, а она пусть вызывает func1 и func2 у себя внутри:

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

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

elem.onclick **=** func3; //тут привязывается только функция func3

//func3 вызывает func1 и func2:

function func3() {

func1();

func2();

}

function func1() {

alert('1');

}

function func2() {

alert('2');

}

Как вы видите, этот недостаток не слишком существенный и его легко обойти. Только что вводится лишняя функция, что немного неудобно.

Однако, есть еще один недостаток - мы не можем легко отвязать от onclick, к примеру, только функцию func1, оставив func2 привязанным. Можно, конечно же, накрутить большие конструкции кода, однако это не нужно, если пользоваться еще более продвинутым способом привязать событие - через addEventListener. Давайте посмотрим, как с работать с этой функцией:

**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 таких проблем, как в предыдущем случае у нас не возникнет - 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](http://code.mu/javascript/dom/getElementsByTagName.html) 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);

}