**Введение в браузерные события**

Для реакции на действия посетителя и внутреннего взаимодействия скриптов существуют *события*.

*Событие* – это сигнал от браузера о том, что что-то произошло. Существует много видов событий. Посмотрим список самых часто используемых, пока просто для ознакомления:

**События мыши:**

* click – происходит, когда кликнули на элемент левой кнопкой мыши
* contextmenu – происходит, когда кликнули на элемент правой кнопкой мыши
* mouseover – возникает, когда на элемент наводится мышь
* mousedown и mouseup – когда кнопку мыши нажали или отжали
* mousemove – при движении мыши

**События на элементах управления:**

* submit – посетитель отправил форму <form>
* focus – посетитель фокусируется на элементе, например нажимает на <input>

**Клавиатурные события:**

* keydown – когда посетитель нажимает клавишу
* keyup – когда посетитель отпускает клавишу

**События документа:**

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

**События CSS:**

* transitionend – когда CSS-анимация завершена.

Также есть и много других событий.

**[Назначение обработчиков событий](http://learn.javascript.ru/introduction-browser-events" \l "naznachenie-obrabotchikov-sobytiy)**

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

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

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

**[Использование атрибута HTML](http://learn.javascript.ru/introduction-browser-events" \l "ispolzovanie-atributa-html)**

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

Например, чтобы прикрепить click-событие к input кнопке, можно присвоить обработчик onclick, вот так:

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

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

В действии:

Обратите внимание, для содержимого атрибута onclick используются *одинарные кавычки*, так как сам атрибут находится в двойных.

Частая ошибка новичков в том, что они забывают, что код находится внутри атрибута. Запись вида onclick="alert("Клик!")", с двойными кавычками внутри, не будет работать. Если вам действительно нужно использовать именно двойные кавычки, то это можно сделать, заменив их на &quot;, то есть так: onclick="alert(&quot;Клик!&quot;)".

Однако, обычно этого не требуется, так как прямо в разметке пишутся только очень простые обработчики. Если нужно сделать что-то сложное, то имеет смысл описать это в функции, и в обработчике вызвать уже её.

Следующий пример по клику запускает функцию countRabbits().

<!DOCTYPE HTML>

<html>

<head>

<meta charset="utf-8">

<script>

function countRabbits() {

for(var i=1; i<=3; i++) {

alert("Кролик номер " + i);

}

}

</script>

</head>

<body>

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

</body>

</html>

Как мы помним, атрибут HTML-тега не чувствителен к регистру, поэтому ONCLICK будет работать так же, как onClick или onCLICK… Но, как правило, атрибуты пишут в нижнем регистре: onclick.

**[Использование свойства DOM-объекта](http://learn.javascript.ru/introduction-browser-events" \l "ispolzovanie-svoystva-dom-obekta)**

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

Пример установки обработчика click:

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

<script>

elem.onclick = function() {

alert( 'Спасибо' );

};

</script>

Если обработчик задан через атрибут, то браузер читает HTML-разметку, создаёт новую функцию из содержимого атрибута и записывает в свойство onclick.

**Этот способ, по сути, аналогичен предыдущему.**

Обработчик хранится именно в DOM-свойстве, а атрибут – лишь один из способов его инициализации.

Эти два примера кода работают одинаково:

1. Только HTML:

<input type="button" onclick="alert('Клик!')" value="Кнопка"/>

1. HTML + JS:

<input type="button" id="button" value="Кнопка" />

<script>

button.onclick = function() {

alert( 'Клик!' );

};

</script>

**Так как DOM-свойство onclick, в итоге, одно, то назначить более одного обработчика так нельзя.**

В примере ниже назначение через JavaScript перезапишет обработчик из атрибута:

<input type="button" id="elem" onclick="alert('До')" value="Нажми меня" />

<script>

elem.onclick = function() { // перезапишет существующий обработчик

alert( 'После' ); // выведется только это

};

</script>

Кстати, обработчиком можно назначить и уже существующую функцию:

function sayThanks() {

alert( 'Спасибо!' );

}

elem.onclick = sayThanks;

Если обработчик надоел – его всегда можно убрать назначением elem.onclick = null.

**[Доступ к элементу через this](http://learn.javascript.ru/introduction-browser-events" \l "dostup-k-elementu-cherez-this)**

Внутри обработчика события this ссылается на текущий элемент, то есть на тот, на котором он сработал.

Это можно использовать, чтобы получить свойства или изменить элемент.

В коде ниже button выводит свое содержимое, используя this.innerHTML:

<button onclick="alert(this.innerHTML)">Нажми меня</button>

В действии: Нажми меня

**[Частые ошибки](http://learn.javascript.ru/introduction-browser-events" \l "chastye-oshibki)**

Если вы только начинаете работать с событиями – обратите внимание на следующие особенности.

**Функция должна быть присвоена как sayThanks, а не sayThanks().**

button.onclick = sayThanks;

Если добавить скобки, то sayThanks() – будет уже *результат* выполнения функции (а так как в ней нет return, то в onclick попадёт undefined). Нам же нужна именно функция.

…А вот в разметке как раз скобки нужны:

<input type="button" id="button" onclick="sayThanks()" />

Это различие просто объяснить. При создании обработчика браузером из атрибута, он автоматически создает функцию из его содержимого. Поэтому последний пример – фактически то же самое, что:

button.onclick = function() {

sayThanks(); // содержимое атрибута

};

**Используйте именно функции, а не строки.**

Назначение обработчика строкой elem.onclick = "alert(1)" можно иногда увидеть в древнем коде. Это будет работать, но не рекомендуется, могут быть проблемы при сжатии JavaScript. Да и вообще, передавать код в виде строки по меньшей мере странно в языке, который поддерживает Function Expressions. Это возможно лишь по соображениям совместимости, не делайте так.

**Не используйте setAttribute.**

Такой вызов работать не будет:

// при нажатии на body будут ошибки

// потому что при назначении в атрибут функция будет преобразована в строку

document.body.setAttribute('onclick', function() { alert(1) });

**Регистр DOM-свойства имеет значение.**

При назначении через DOM нужно использовать свойство onclick, а не ONCLICK.

**[Недостаток назначения через свойство](http://learn.javascript.ru/introduction-browser-events" \l "nedostatok-naznacheniya-cherez-svoystvo)**

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

Например, одна часть кода хочет при клике на кнопку делать ее подсвеченной, а другая – выдавать сообщение. Нужно в разных местах два обработчика повесить.

При этом новый обработчик будет затирать предыдущий. Например, следующий код на самом деле назначает один обработчик – последний:

input.onclick = function() { alert(1); }

// ...

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

Разработчики стандартов достаточно давно это поняли и предложили альтернативный способ назначения обработчиков при помощи специальных методов, которые свободны от указанного недостатка.

**[addEventListener и removeEventListener](http://learn.javascript.ru/introduction-browser-events" \l "addeventlistener-i-removeeventlistener)**

Методы addEventListener и removeEventListener являются современным способом назначить или удалить обработчик, и при этом позволяют использовать сколько угодно любых обработчиков.

Назначение обработчика осуществляется вызовом addEventListener с тремя аргументами:

element.addEventListener(event, handler[, phase]);

**event**

Имя события, например click

**handler**

Ссылка на функцию, которую надо поставить обработчиком.

**phase**

Необязательный аргумент, «фаза», на которой обработчик должен сработать. Этот аргумент редко нужен, мы его рассмотрим позже.

Удаление обработчика осуществляется вызовом removeEventListener:

// передать те же аргументы, что были у addEventListener

element.removeEventListener(event, handler[, phase]);

**Удаление требует именно ту же функцию**

Для удаления нужно передать именно ту функцию-обработчик которая была назначена.

Вот так removeEventListener не сработает:

elem.addEventListener( "click" , function() {alert('Спасибо!')});

// ....

elem.removeEventListener( "click", function() {alert('Спасибо!')});

В removeEventListener передана не та же функция, а другая, с одинаковым кодом, но это не важно.

Вот так правильно:

function handler() {

alert( 'Спасибо!' );

}

input.addEventListener("click", handler);

// ....

input.removeEventListener("click", handler);

Обратим внимание – если функцию не сохранить где-либо, а просто передать в addEventListener, как в предыдущем коде, то потом получить её обратно, чтобы снять обработчик, будет невозможно. Нет метода, который позволяет считать обработчики событий, назначенные через addEventListener.

Метод addEventListener позволяет добавлять несколько обработчиков на одно событие одного элемента, например:

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

<script>

function handler1() {

alert('Спасибо!');

};

function handler2() {

alert('Спасибо ещё раз!');

}

elem.onclick = function() { alert("Привет"); };

elem.addEventListener("click", handler1); // Спасибо!

elem.addEventListener("click", handler2); // Спасибо ещё раз!

</script>

Как видно из примера выше, можно одновременно назначать обработчики и через DOM-свойство и через addEventListener. Однако, во избежание путаницы, рекомендуется выбрать один способ.

**addEventListener работает всегда, а DOM-свойство – нет**

У специальных методов есть ещё одно преимущество перед DOM-свойствами.

Есть некоторые события, которые нельзя назначить через DOM-свойство, но можно через addEventListener.

Например, таково событие transitionend, то есть окончание CSS-анимации. В большинстве браузеров оно требует назначения через addEventListener.

Вы можете проверить это, запустив код в примере ниже. Как правило, сработает лишь второй обработчик, но не первый.

<style>

button {

transition: width 1s;

width: 100px;

}

.wide {

width: 300px;

}

</style>

<button id="elem" onclick="this.classList.toggle('wide');">

Нажми меня

</button>

<script>

elem.ontransitionend = function() {

alert( "ontransitionend" ); // не сработает

};

elem.addEventListener("transitionend", function() {

alert( "addEventListener" ); // сработает по окончании анимации

});

</script>

**[Отличия IE8-](http://learn.javascript.ru/introduction-browser-events" \l "otlichiya-ie8)**

При работе с событиями в IE8- есть много отличий. Как правило, они формальны – некое свойство или метод называются по-другому. Начиная с версии 9, также работают и стандартные свойства и методы.

**В IE8- вместо addEventListener/removeEventListener используются свои методы.**

Назначение обработчика осуществляется вызовом attachEvent:

element.attachEvent("on" + event, handler);

Удаление обработчика – вызовом detachEvent:

element.detachEvent("on" + event, handler);

Например:

function handler() {

alert( 'Спасибо!' );

}

button.attachEvent("onclick", handler) // Назначение обработчика

// ....

button.detachEvent("onclick", handler) // Удаление обработчика

Как видите, почти то же самое, только событие должно включать префикс on.

**У обработчиков, назначенных с attachEvent, нет this**

Обработчики, назначенные с attachEvent не получают this!

Это важная особенность и подводный камень старых IE.

Чтобы ваш код работал в старом IE, нужно либо использовать DOM-свойства, то есть onclick, либо подключить полифилл для современных методов, например [такой](https://gist.github.com/jonathantneal/3748027) или с сервиса [polyfill.io](http://polyfill.webservices.ft.com/v1/docs/features/) или какой-то другой.

**[Итого](http://learn.javascript.ru/introduction-browser-events" \l "itogo)**

Есть три способа назначения обработчиков событий:

1. Атрибут HTML: onclick="...".
2. Свойство: elem.onclick = function.
3. Специальные методы:

* Современные: elem.addEventListener( событие, handler[, phase]), удаление через removeEventListener.
* Для старых IE8-: elem.attachEvent( on+событие, handler ), удаление через detachEvent.

Сравнение addEventListener и onclick:

**Достоинства**

* Некоторые события можно назначить только через addEventListener.
* Метод addEventListener позволяет назначить много обработчиков на одно событие.

**Недостатки**

* Обработчик, назначенный через onclick, проще удалить или заменить.
* Метод onclick кросс-браузерный.

Этим введением мы только открываем работу с событиями, но вы уже можете решать разнообразные задачи с их использованием.