DOM. События

Оглавление

[Операторы и операнды 0](#_Toc447573385)

[Приоритет операций 1](#_Toc447573386)

[Побитовые операторы 6](#_Toc447573387)

[Используемая литература 7](#_Toc447573388)

# 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**

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**

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.

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 видаonclick="alert("Клик!")", с двойными кавычки 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**

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.

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> |

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**

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]); |

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.

# Объект события

Чтобы хорошо обработать событие, недостаточно знать о том, что это – «клик» или «нажатие клавиши». Могут понадобиться детали: координаты курсора, введённый символ и другие, в зависимости от события.

Детали произошедшего браузер записывает в «объект события», который передаётся первым аргументом в обработчик.

**Свойства объекта события**

Пример ниже демонстрирует использование объекта события:

|  |
| --- |
| <input type="button" value="Нажми меня" id="elem">  <script>  elem.onclick = function(event) {  // вывести тип события, элемент и координаты клика  alert(event.type + " на " + event.currentTarget);  alert(event.clientX + ":" + event.clientY);  }  </script> |

Свойства объекта event:

**event.type**

Тип события, в данном случае click

**event.currentTarget**

Элемент, на котором сработал обработчик. Значение – в точности такое же, как и у this, но бывают ситуации, когда обработчик является методом объекта и его this при помощи bind привязан к этому объекту, тогда мы можем использовать event.currentTarget.

**event.clientX / event.clientY**

Координаты курсора в момент клика (относительно окна)

Есть также и ряд других свойств, в зависимости от событий, которые мы разберём в дальнейших главах, когда будем подробно знакомиться с событиями мыши, клавиатуры и так далее.

# Всплытие и перехват

Этот обработчик для <div> сработает, если вы кликните по вложенному тегу <em> или <code>:

|  |
| --- |
| <div onclick="alert('Обработчик для Div сработал!')">  <em>Кликните на <code>EM</code>, сработает обработчик на <code>DIV</code></em>  </div> |

Почему же сработал обработчик на <div>, если клик произошёл на<em>?

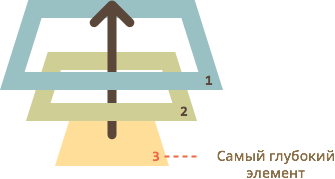
Основной принцип всплытия:

**При наступлении события обработчики сначала срабатывают на самом вложенном элементе, затем на его родителе, затем выше и так далее, вверх по цепочке вложенности.**

Например, есть 3 вложенных элемента FORM > DIV > P, с обработчиком на каждом:

|  |
| --- |
| <style>  body \* {  margin: 10px;  border: 1px solid blue;  }  </style>  <form onclick="alert('form')">FORM  <div onclick="alert('div')">DIV  <p onclick="alert('p')">P</p>  </div>  </form> |

Всплытие гарантирует, что клик по внутреннему <p> вызовет обработчик onclick (если есть) сначала на самом <p>, затем на элементе <div> далее на элементе <form>, и так далее вверх по цепочке родителей до самого document.



Поэтому если в примере выше кликнуть на P, то последовательно выведутся alert: p → div → form.

Этот процесс называется *всплытием*, потому что события «всплывают» от внутреннего элемента вверх через родителей, подобно тому, как всплывает пузырек воздуха в воде.

На каком бы элементе мы ни поймали событие, всегда можно узнать, где конкретно оно произошло.

**Самый глубокий элемент, который вызывает событие, называется *«целевым»* или*«исходным»* элементом и доступен как event.target.**

Отличия от this (=event.currentTarget):

* event.target – это **исходный элемент**, на котором произошло событие, в процессе всплытия он неизменен.
* this – это **текущий элемент**, до которого дошло всплытие, на нём сейчас выполняется обработчик.

Например, если стоит только один обработчик form.onclick, то он «поймает» все клики внутри формы. Где бы ни был клик внутри – он всплывёт до элемента <form>, на котором сработает обработчик.

При этом:

* this (=event.currentTarget) всегда будет сама форма, так как обработчик сработал на ней.
* event.target будет содержать ссылку на конкретный элемент внутри формы, самый вложенный, на котором произошёл клик.

Возможна и ситуация, когда event.target и this – один и тот же элемент, например если в форме нет других тегов и клик был на самом элементе <form>.

**Прекращение всплытия**

Всплытие идёт прямо наверх. Обычно событие будет всплывать наверх и наверх, до элемента <html>, а затем до document, а иногда даже до window, вызывая все обработчики на своем пути.

Но любой промежуточный обработчик может решить, что событие полностью обработано, и остановить всплытие.

Для остановки всплытия нужно вызвать метод event.stopPropagation().

Например, здесь при клике на кнопку обработчик body.onclick не сработает:

|  |
| --- |
| <body onclick="alert('сюда обработка не дойдёт')">  <button onclick="event.stopPropagation()">Кликни меня</button>  </body> |

# Действия браузера по умолчанию

Многие события автоматически влекут за собой действие браузера.

Например:

* Клик по ссылке инициирует переход на новый URL.
* Нажатие на кнопку «отправить» в форме – отсылку ее на сервер.
* Двойной клик на тексте – инициирует его выделение.

Если мы обрабатываем событие в JavaScript, то зачастую такое действие браузера нам не нужно. К счастью, его можно отменить.

**Отмена действия браузера**

Есть два способа отменить действие браузера:

* **Основной способ – это воспользоваться объектом события. Для отмены действия браузера существует стандартный метод event.preventDefault().**
* Если же обработчик назначен через onсобытие (не через addEventListener), то можно просто вернуть false из обработчика.

В следующем примере при клике по ссылке переход не произойдет:

|  |
| --- |
| <a href="/" onclick="return false">Нажми здесь</a>  или  <a href="/" onclick="event.preventDefault()">здесь</a> |

Действий браузера по умолчанию достаточно много.

Вот некоторые примеры событий, которые вызывают действие браузера:

* mousedown – нажатие кнопкой мыши в то время как курсор находится на тексте начинает его выделение.
* click на <input type="checkbox"> – ставит или убирает галочку.
* submit – при нажатии на <input type="submit"> в форме данные отправляются на сервер.
* wheel – движение колёсика мыши инициирует прокрутку.
* keydown – при нажатии клавиши в поле ввода появляется символ.
* contextmenu – при правом клике показывается контекстное меню браузера.
* …

Все эти действия можно отменить, если мы хотим обработать событие исключительно при помощи JavaScript.

# Домашнее задание

* Используя результаты домашнего задания к предыдущему уроку, выполнить следующее. При щелчке мышью на любую клетку доски – необходимо писать ее адрес в произвольное место страницы, например, в div. Адрес должен извлекаться в «шахматном» формате. Например: A1, G6 и тп. При этом, ячейка, на которую нажали должна помечаться произвольным образом, например, выделением рамки или другим цветом. При выделении другой ячейки, предыдущая должна возвращаться к первоначальному виду.
* Научиться обрабатывать стрелки клавиатуры таким образом, чтобы активную ячейку из предыдущего пункта можно было перемещать по доске. Если ячейка выходит за границы таблицы – она должна появиться с другой стороны. При перемещении ячейки, так же должен извлекаться ее адрес.
* \* Добавить возможность удалять фигуры с доски и возвращать их на прежнее место. Таким образом, необходимо добавить вверху и внизу таблицы область, для «уничтоженных» фигур. При нажатии на любую фигуру на доске, она должна становиться «уничтоженной», т.е. удаляться с доски и перемещаться в соответствующую область (черные в одну сторону, белые – в другую). При нажатии на «уничтоженную» фигуру она должна вернуться на прежнее место.

# Практика

Продолжим работать над интерфейсом адресной книги. Добавим нашему приложению интерактивности. А именно, добавим вызов разработанных ранее методов по определенным событиям. При нажатии на какой-нибудь контакт должна показываться детальная информация пользователя (телефон и пр.).

Так же, напротив каждого пользователя добавим крестик, при нажатии на который должен вызываться метод deleteUser, реализованный в рамках предыдущих домашних заданий.

Кроме того, следует добавить текстовое поле, навешать на него обработчик события keydown, в котором запускать функцию search, разработанную на прошлых занятия.

И, конечно, необходимо добавить кнопку при нажатии на которую будет вызываться метод добавления пользователя addUser.

Пример реализации описанных задач можно найти в материалах к уроку.

# Используемая литература

* [Mozilla Developer Network](https://developer.mozilla.org/)
* [MSDN](http://msdn.microsoft.com/)
* [Safari Developer Library](https://developer.apple.com/library/safari/navigation/index.html)
* [Современный учебник JavaScript](http://learn.javascript.ru)