**Генерация событий на элементах**

Можно не только назначать обработчики на события, но и генерировать их самому.

Мы будем использовать это позже для реализации компонентной архитектуры, при которой элемент, представляющий собой, к примеру, меню, генерирует события, к этому меню относящиеся – select (выбран пункт меню) или open (меню раскрыто), и другие.

Кроме того, события можно генерировать для целей автоматического тестирования.

**[Конструктор Event](http://learn.javascript.ru/dispatch-events" \l "konstruktor-event)**

Вначале рассмотрим современный способ генерации событий, по стандарту [DOM 4](http://www.w3.org/TR/dom/#introduction-to-dom-events). Он поддерживается всеми браузерами, кроме IE11-. А далее рассмотрим устаревшие варианты, поддерживаемые IE.

Объект события в нём создаётся при помощи встроенного конструктора [Event](http://www.w3.org/TR/dom/" \l "event).

Синтаксис:

var event = new Event(тип события[, флаги]);

Где:

* *Тип события* – может быть как своим, так и встроенным, к примеру "click".
* *Флаги* – объект вида { bubbles: true/false, cancelable: true/false }, где свойство bubbles указывает, всплывает ли событие, а cancelable – можно ли отменить действие по умолчанию.

Флаги по умолчанию: {bubbles: false, cancelable: false}.

**[Метод dispatchEvent](http://learn.javascript.ru/dispatch-events" \l "metod-dispatchevent)**

Затем, чтобы инициировать событие, запускается elem.dispatchEvent(event).

При этом событие срабатывает наравне с браузерными, то есть обычные браузерные обработчики на него отреагируют. Если при создании указан флаг bubbles, то оно будет всплывать.

При просмотре примера ниже обработчик onclick на кнопке сработает сам по себе, событие генерируется скриптом:

<button id="elem" onclick="alert('Клик');">Автоклик</button>

<script>

var event = new Event("click");

elem.dispatchEvent(event);

</script>

**[Отмена действия по умолчанию](http://learn.javascript.ru/dispatch-events" \l "otmena-deystviya-po-umolchaniyu)**

На сгенерированном событии, как и на встроенном браузерном, обработчик может вызвать метод event.preventDefault(). Тогда dispatchEvent возвратит false.

Остановимся здесь подробнее. Обычно вызов preventDefault() предотвращает действие браузера. В случае, если событие придумано нами, имеет нестандартное имя – никакого действия браузера, конечно, нет.

Но код, который генерирует событие, может предусматривать какие-то ещё действия после dispatchEvent.

Вызов event.preventDefault() является возможностью для обработчика события сообщить в сгенерировавший событие код, что эти действия продолжать не надо.

В примере ниже есть функция hide(), которая при вызове генерирует событие hide на элементе #rabbit, уведомляя всех интересующихся, что кролик собирается спрятаться.

Любой обработчик может узнать об этом, подписавшись на событие через rabbit.addEventListener('hide',...)и, при желании, отменить действие по умолчанию через event.preventDefault(). Тогда кролик не исчезнет:

<pre id="rabbit">

|\ /|

\|\_|/

/. .\

=\\_Y\_/=

{>o<}

</pre>

<script>

function hide() {

var event = new Event("hide", {

cancelable: true

});

if (!rabbit.dispatchEvent(event)) {

alert( 'действие отменено обработчиком' );

} else {

rabbit.hidden = true;

}

}

rabbit.addEventListener('hide', function(event) {

if (confirm("Вызвать preventDefault?")) {

event.preventDefault();

}

});

// прячемся через 2 секунды

setTimeout(hide, 2000);

</script>

**Как отличить реальное нажатие от скриптового?**

В целях безопасности иногда хорошо бы знать – инициировано ли действие посетителем или это кликнул скрипт.

Единственный способ, которым код может отличить реальное нажатие от программного, является проверка свойства event.isTrusted.

Оно на момент написания статьи поддерживается IE и Firefox и равно true, если посетитель кликнул сам, и всегда false – если событие инициировал скрипт.

**[Другие свойства событий](http://learn.javascript.ru/dispatch-events" \l "drugie-svoystva-sobytiy)**

При создании события браузер автоматически ставит следующие свойства:

* isTrusted: false – означает, что событие сгенерировано скриптом, это свойство изменить невозможно.
* target: null – это свойство ставится автоматически позже при dispatchEvent.
* type: тип события – первый аргумент new Event.
* bubbles, cancelable – по второму аргументу new Event.

Другие свойства события, если они нужны, например координаты для события мыши – можно присвоить в объект события позже, например:

var event = new Event("click", {bubbles: true, cancelable: false});

event.clientX = 100;

event.clientY = 100;

**[Пример со всплытием](http://learn.javascript.ru/dispatch-events" \l "primer-so-vsplytiem)**

Сгенерируем совершенно новое событие "hello" и поймаем его на document.

Всё, что для этого нужно – это флаг bubbles:

<h1 id="elem">Привет от скрипта!</h1>

<script>

document.addEventListener("hello", function(event) { // (1)

alert("Привет");

event.preventDefault(); // (2)

}, false);

var event = new Event("hello", {bubbles: true, cancelable: true}); // (3)

if (elem.dispatchEvent(event) === false) {

alert('Событие было отменено preventDefault');

}

</script>

Обратите внимание:

1. Обработчик события hello стоит на document. Мы его поймаем на всплытии.
2. Вызов event.preventDefault() приведёт к тому, что dispatchEvent вернёт false.
3. Чтобы событие всплывало и его можно было отменить, указан второй аргумент new Event.

Никакой разницы между встроенными событиями (click) и своими (hello) здесь нет, их можно сгенерировать и запустить совершенно одинаково.

**[Конструкторы MouseEvent, KeyboardEvent и другие](http://learn.javascript.ru/dispatch-events" \l "konstruktory-mouseevent-keyboardevent-i-drugie)**

Для некоторых конкретных типов событий есть свои, специфические, конструкторы.

Вот список конструкторов для различных событий интерфейса которые можно найти в спецификации [UI Event](http://www.w3.org/TR/uievents/):

* UIEvent
* FocusEvent
* MouseEvent
* WheelEvent
* KeyboardEvent
* CompositionEvent

Вместо new Event("click") можно вызвать new MouseEvent("click").

**Специфический конструктор позволяет указать стандартные свойства для данного типа события.**

Например, clientX/clientY для события мыши:

var e = new MouseEvent("click", {

bubbles: true,

cancelable: true,

clientX: 100,

clientY: 100

});

alert( e.clientX ); // 100

Это нельзя было бы сделать с обычным конструктором Event:

var e = new Event("click", {

bubbles: true,

cancelable: true,

clientX: 100,

clientY: 100

});

alert( e.clientX ); // undefined, свойство не присвоено!

Обычный конструктор Event не знает про «мышиные» свойства, поэтому их игнорирует.

Впрочем, использование конкретного конструктора не является обязательным, можно обойтись Event, а свойства записать в объект отдельно, после конструктора. Здесь это скорее вопрос удобства и желания следовать правилам. События, которые генерирует браузер, всегда имеют правильный тип.

Полный список свойств по типам событий вы найдёте в спецификации, например для MouseEvent: [MouseEvent Constructor](http://www.w3.org/TR/uievents/" \l "constructor-mouseevent).

**[Свои события](http://learn.javascript.ru/dispatch-events" \l "svoi-sobytiya)**

Для генерации своих, нестандартных, событий, хоть и можно использовать конструктор Event, но существует и специфический конструктор [CustomEvent](http://www.w3.org/TR/dom/" \l "customevent).

Технически, он абсолютно идентичен Event, кроме небольшой детали: у второго аргумента-объекта есть дополнительное свойство detail, в котором можно указывать информацию для передачи в событие.

Например:

<h1 id="elem">Привет для Васи!</h1>

<script>

elem.addEventListener("hello", function(event) {

alert( event.detail.name );

}, false);

var event = new CustomEvent("hello", {

detail: { name: "Вася" }

});

elem.dispatchEvent(event);

</script>

Надо сказать, что никто не мешает и в обычное Event записать любые свойства. Но CustomEvent более явно говорит, что событие не встроенное, а своё, и выделяет отдельно «информационное» поле detail, в которое можно записать что угодно без конфликта со стандартными свойствами объекта.

**[Старое API для IE9+](http://learn.javascript.ru/dispatch-events" \l "staroe-api-dlya-ie9)**

Способ генерации событий, описанный выше, не поддерживается в IE11-, там нужен другой, более старый способ, описанный в стандарте [DOM 3 Events](http://www.w3.org/TR/DOM-Level-3-Events).

В нём была предусмотрена [иерархия событий](http://www.w3.org/TR/DOM-Level-3-Events/#event-interfaces), с различными методами инициализации.

Она поддерживается как современными браузерами, так и IE9+. Там используется немного другой синтаксис, но по возможностям – всё то же самое, что и в современном стандарте.

Можно использовать этот немного устаревший способ, если нужно поддерживать IE9+. Далее мы на его основе создадим полифилл.

Объект события создаётся вызовом document.createEvent:

var event = document.createEvent(eventInterface);

Аргументы:

* eventInterface – это тип события, например MouseEvent, FocusEvent, KeyboardEvent. В [секции 5 DOM 3 Events](http://www.w3.org/TR/DOM-Level-3-Events/#events-module) есть подробный список, какое событие к какому интерфейсу относится.

На практике можно всегда использовать самый общий интерфейс: document.createEvent("Event").

Далее событие нужно инициализировать:

event.initEvent(type, boolean bubbles, boolean cancelable);

Аргументы:

* type – тип события, например "click".
* bubbles – всплывает ли событие.
* cancelable – можно ли отменить событие.

Эти два кода аналогичны:

// современный стандарт

var event = new Event("click", {

bubbles: true,

cancelable: true

});

// старый стандарт

var event = document.createEvent("Event");

event.initEvent("click", true, true);

Единственная разница – старый стандарт поддерживается IE9+.

Этот пример с событием hello будет работать во всех браузерах, кроме IE8-:

<h1 id="elem">Привет от скрипта!</h1>

<script>

document.addEventListener("hello", function(event) {

alert( "Привет" );

event.preventDefault();

}, false);

var event = document.createEvent("Event");

event.initEvent("hello", true, true);

if (elem.dispatchEvent(event) === false) {

alert( 'Событие было отменено preventDefault' );

}

</script>

**initMouseEvent, initKeyboardEvent и другие…**

У конкретных типов событий, например MouseEvent, KeyboardEvent, есть методы, которые позволяют указать стандартные свойства.

Они называются по аналогии: initMouseEvent, initKeyboardEvent.

Их можно использовать вместо базового initEvent, если хочется, чтобы свойства событий соответствовали встроенным браузерным.

Выглядят они немного страшновато, например (взято из [спецификации](http://www.w3.org/TR/DOM-Level-3-Events/#idl-interface-MouseEvent-initializers)):

void initMouseEvent(

DOMString typeArg, // тип

boolean bubblesArg, // всплывает?

boolean cancelableArg, // можно отменить?

AbstractView ? viewArg, // объект window, null означает текущее окно

long detailArg, // свойство detail и другие...

long screenXArg,

long screenYArg,

long clientXArg,

long clientYArg,

boolean ctrlKeyArg,

boolean altKeyArg,

boolean shiftKeyArg,

boolean metaKeyArg,

unsigned short buttonArg,

EventTarget ? relatedTargetArg);

};

Для инициализации мышиного события нужно обязательно указать *все* аргументы, например:

<button id="elem">Автоклик</button>

<script>

elem.onclick = function(e) {

alert( 'Клик на координатах ' + e.clientX + ':' + e.clientY );

};

var event = document.createEvent("MouseEvent");

event.initMouseEvent("click", true, true, null, 0, 0, 0, 100, 100, true, true, true, null, 1, null);

elem.dispatchEvent(event);

</script>

Браузер, по стандарту, может сгенерировать отсутствующие свойства самостоятельно, например pageX, но это нужно проверять в конкретных случаях, иногда это не работает или работает некорректно, так что лучше указать все.

**[Полифилл CustomEvent](http://learn.javascript.ru/dispatch-events" \l "polifill-customevent)**

Для поддержки CustomEvent в IE9+ можно сделать небольшой полифилл:

try {

new CustomEvent("IE has CustomEvent, but doesn't support constructor");

} catch (e) {

window.CustomEvent = function(event, params) {

var evt;

params = params || {

bubbles: false,

cancelable: false,

detail: undefined

};

evt = document.createEvent("CustomEvent");

evt.initCustomEvent(event, params.bubbles, params.cancelable, params.detail);

return evt;

};

CustomEvent.prototype = Object.create(window.Event.prototype);

}

Здесь мы сначала проверяем – в IE9-11 есть CustomEvent, но его нельзя создать через new, будет ошибка. В этом случае заменяем браузерную реализацию на свою, совместимую.

**[Антистандарт: IE8-](http://learn.javascript.ru/dispatch-events" \l "antistandart-ie8)**

В совсем старом IE были «свои» методы document.createEventObject() и elem.fireEvent().

Пример с ними для IE8:

<button id="elem">Автоклик</button>

<script>

document.body.onclick = function() {

alert( "Клик, event.type=" + event.type );

return false;

};

var event = document.createEventObject();

if (!elem.fireEvent("onclick", event)) {

alert( 'Событие было отменено' );

}

</script>

**При помощи fireEvent можно сгенерировать только встроенные события.**

Если указать "hello" вместо "onclick" в примере выше – будет ошибка.

Параметры bubbles и cancelable настраивать нельзя, браузер использует стандартные для данного типа событий.

Существуют полифиллы для генерации произвольных событий и для IE8-, но они, по сути, полностью подменяют встроенную систему обработки событий браузером. И кода это требует тоже достаточно много.

Альтернатива – фреймворк, например jQuery, который также реализует свою мощную систему работы с событиями, доступную через методы jQuery.

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

* Все браузеры, кроме IE9-11, позволяют генерировать любые события, следуя стандарту DOM4.
* В IE9+ поддерживается более старый стандарт, можно легко сделать полифилл, например для CustomEvent он рассмотрен в этой главе.
* IE8- может генерировать только встроенные события.

Несмотря на техническую возможность генерировать встроенные браузерные события типа click или keydown – пользоваться ей стоит с большой осторожностью.

В 98% случаев, когда разработчик начинающего или среднего уровня хочет сгенерировать *встроенное* событие – это вызвано «кривой» архитектурой кода, и взаимодействие нужно на уровне выше.

Как правило события имеет смысл генерировать:

* Либо как явный и грубый хак, чтобы заставить работать сторонние библиотеки, в которых не предусмотрены другие средства взаимодействия.
* Либо для автоматического тестирования, чтобы скриптом «нажать на кнопку» и посмотреть, произошло ли нужное действие.
* Либо при создании своих «элементов интерфейса». Например, никто не мешает при помощи JavaScript создать из <div class="calendar"> красивый календарь и генерировать на нём событие change при выборе даты. Эту тему мы разовьём позже.