

Раздел

[Введение в события](#)

Навигация по уроку

Конструктор Event

Метод `dispatchEvent`

Пример всплытия

MouseEvent, KeyboardEvent и другие

Пользовательские события

`event.preventDefault()`Вложенные события
обрабатываются синхронно


Итого

Комментарии

Поделиться



Редактировать на GitHub

[🏠](#) → [Браузер: документ, события, интерфейсы](#)
→ [Введение в события](#) 21-го июня 2020

Генерация пользовательских событий

Можно не только назначать обработчики, но и генерировать события из JavaScript-кода.

Пользовательские события могут быть использованы при создании графических компонентов. Например, корневой элемент нашего меню, реализованного при помощи JavaScript, может генерировать события, относящиеся к этому меню: `open` (меню раскрыто), `select` (выбран пункт меню) и т.п. А другой код может слушать эти события и узнавать, что происходит с меню.

Можно генерировать не только совершенно новые, придуманные нами события, но и встроенные, такие как `click`, `mousedown` и другие. Это бывает полезно для автоматического тестирования.

Конструктор Event

Встроенные классы для событий формируют иерархию аналогично классам для DOM-элементов. Её корнем является встроенный класс [Event](#).

Событие встроенного класса `Event` можно создать так:

```
1 let event = new Event(type[, options]);
```

Где:

- `type` – тип события, строка, например `"click"` или же любой придуманный нами – `"my-event"`.
- `options` – объект с тремя необязательными свойствами:
 - `bubbles`: `true/false` – если `true`, тогда событие всплывает.
 - `cancelable`: `true/false` – если `true`, тогда можно отменить действие по умолчанию. Позже мы разберём, что это значит для пользовательских событий.
 - `composed`: `true/false` – если `true`, тогда событие будет всплывать наружу за пределы Shadow DOM. Позже мы разберём это в [разделе Веб-компоненты](#).

По умолчанию все три свойства установлены в **false**: `{bubbles: false, cancelable: false, composed: false}`.

Метод `dispatchEvent`

После того, как объект события создан, мы должны запустить его на элементе, вызвав метод `elem.dispatchEvent(event)`.

Затем обработчики отреагируют на него, как будто это обычное браузерное событие. Если при создании указан флаг `bubbles`, то оно будет всплывать.

В примере ниже событие `click` инициируется JavaScript-кодом так, как будто кликнули по кнопке:

```
1 <button id="elem" onclick="alert('Клик!');">Автоклик</b>  
2  
3 <script>  
4   let event = new Event("click");  
5   elem.dispatchEvent(event);  
6 </script>
```

Раздел

[Введение в события](#)

Навигация по уроку

Конструктор Event

Метод `dispatchEvent`

Пример всплытия

MouseEvent, KeyboardEvent и другие

Пользовательские события

`event.preventDefault()`

Вложенные события
обрабатываются синхронно

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



event.isTrusted

Можно легко отличить «настоящее» событие от сгенерированного кодом.

Свойство `event.isTrusted` принимает значение `true` для событий, порожденных реальными действиями пользователя, и `false` для генерируемых кодом.

Пример всплытия

Мы можем создать всплывающее событие с именем "hello" и поймать его на `document`.

Всё, что нужно сделать – это установить флаг `bubbles` в `true`:

```
1 <h1 id="elem">Привет из кода!</h1>
2
3 <script>
4   // ловим на document...
5   document.addEventListener("hello", function(event) {
6     alert("Привет от " + event.target.tagName); // Прив
7   });
8
9   // ...запуск события на элементе!
10  let event = new Event("hello", {bubbles: true}); // (
11  elem.dispatchEvent(event);
12
13  // обработчик на document работает и выведет сообщен
14
15 </script>
```



Обратите внимание:



1. Мы должны использовать `addEventListener` для наших собственных событий, т.к. `on<event>`-свойства существуют только для встроенных событий, то есть `document.onhello` не работает.
2. Мы обязаны передать флаг `bubbles: true`, иначе наше событие не будет всплывать.

Механизм всплытия идентичен как для встроенного события (`click`), так и для пользовательского события (`hello`). Также одинакова работа фаз всплытия и погружения.

MouseEvent, KeyboardEvent и другие

Для некоторых конкретных типов событий есть свои специфические конструкторы. Вот небольшой список конструкторов для различных событий пользовательского интерфейса, которые можно найти в спецификации [UI Event](#):

- `UIEvent`
- `FocusEvent`
- `MouseEvent`
- `WheelEvent`
- `KeyboardEvent`
- ...

Стоит использовать их вместо `new Event`, если мы хотим создавать такие события. К примеру, `new MouseEvent("click")`.

Специфический конструктор позволяет указать стандартные свойства для данного типа события.

Например, `clientX/clientY` для события мыши:

```
1 let event = new MouseEvent("click", {
2   bubbles: true,
```

Раздел

Введение в события

Навигация по уроку

Конструктор Event

Метод `dispatchEvent`

Пример всплытия

`MouseEvent`, `KeyboardEvent` и другие

Пользовательские события

`event.preventDefault()`

Вложенные события
обрабатываются синхронно

Итого

Комментарии

Поделиться



Редактировать на GitHub



```
3   cancelable: true,
4   clientX: 100,
5   clientY: 100
6 });
7
8 alert(event.clientX); // 100
```

Обратите внимание: этого нельзя было бы сделать с обычным конструктором `Event`.

Давайте проверим:

```
1 let event = new Event("click", {
2   bubbles: true, // только свойства bubbles и cancelable
3   cancelable: true, // работают в конструкторе Event
4   clientX: 100,
5   clientY: 100
6 });
7
8 alert(event.clientX); // undefined, неизвестное свойство
```

Впрочем, использование конкретного конструктора не является обязательным, можно обойтись `Event`, а свойства записать в объект отдельно, после создания, вот так: `event.clientX=100`. 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, [MouseEvent](#).

Пользовательские события

Для генерации событий совершенно новых типов, таких как "hello", следует использовать конструктор `new CustomEvent`. Технически `CustomEvent` абсолютно идентичен `Event` за исключением одной небольшой детали.

У второго аргумента-объекта есть дополнительное свойство `detail`, в котором можно указывать информацию для передачи в событие.

Например:

```
1 <h1 id="elem">Привет для Васи!</h1>
2
3 <script>
4   // дополнительная информация приходит в обработчик в
5   elem.addEventListener("hello", function(event) {
6     alert(event.detail.name);
7   });
8
9   elem.dispatchEvent(new CustomEvent("hello", {
10     detail: { name: "Вася" }
11   }));
12 </script>
```

Свойство `detail` может содержать любые данные. Надо сказать, что никто не мешает и в обычное `new Event` записать любые свойства. Но `CustomEvent` предоставляет специальное поле `detail` во избежание конфликтов с другими свойствами события.

Кроме того, класс события описывает, что это за событие, и если оно не браузерное, а пользовательское, то лучше использовать `CustomEvent`, чтобы явно об этом сказать.

`event.preventDefault()`

Для многих браузерных событий есть «действия по умолчанию», такие как переход по ссылке, выделение и т.п.

Раздел

[Введение в события](#)

Навигация по уроку

Конструктор Event

Метод `dispatchEvent`

Пример всплытия

MouseEvent, KeyboardEvent и другие

Пользовательские события

`event.preventDefault()`

Вложенные события обрабатываются синхронно

Итого

Комментарии

Поделиться



Редактировать на GitHub



Для новых, пользовательских событий браузерных действий, конечно, нет, но код, который генерирует такое событие, может предусматривать какие-то свои действия после события.

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

Тогда вызов `elem.dispatchEvent(event)` возвратит `false`. И код, сгенерировавший событие, узнает, что продолжать не нужно.

Посмотрим практический пример – прячущегося кролика (могло бы быть скрывающееся меню или что-то ещё).

Ниже вы можете видеть кролика `#rabbit` и функцию `hide()`, которая при вызове генерирует на нём событие `"hide"`, уведомляя всех интересующихся, что кролик собирается спрятаться.

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

```
1 <pre id="rabbit">
2   | \   / |
3   \|_  | /
4   / .  . \
5   =\_Y_/ =
6   {>o<}
7 </pre>
8 <button onclick="hide()">Hide()</button>
9
10 <script>
11   // hide() будет вызван автоматически через 2 секунды
12   function hide() {
13     let event = new CustomEvent("hide", {
14       cancelable: true // без этого флага preventDefault
15     });
16     if (!rabbit.dispatchEvent(event)) {
17       alert('Действие отменено обработчиком');
18     } else {
19       rabbit.hidden = true;
20     }
21   }
22
23   rabbit.addEventListener('hide', function(event) {
24     if (confirm("Вызвать preventDefault?")) {
25       event.preventDefault();
26     }
27   });
28 </script>
```

```
| \   / |
| \|_  | /
| / .  . \
| =\_Y_/ =
| {>o<}
```

Hide()

Обратите внимание: событие должно содержать флаг `cancelable: true`. Иначе, вызов `event.preventDefault()` будет проигнорирован.

Вложенные события обрабатываются синхронно

Обычно события обрабатываются асинхронно. То есть, если браузер обрабатывает `onclick` и в процессе этого произойдёт новое событие, то оно ждёт, пока закончится обработка `onclick`.

Исключением является ситуация, когда событие инициировано из обработчика другого события.

Тогда управление сначала переходит в обработчик вложенного события и уже после этого возвращается назад.

Раздел

Введение в события

Навигация по уроку

Конструктор Event

Метод dispatchEvent

Пример всплытия

MouseEvent, KeyboardEvent и другие

Пользовательские события

event.preventDefault()

Вложенные события обрабатываются синхронно

Итого

Комментарии

Поделиться



Редактировать на GitHub

В примере ниже событие `menu-open` обрабатывается синхронно во время обработки `onclick`:



```
1 <button id="menu">Меню (нажми меня)</button>
2
3 <script>
4   menu.onclick = function() {
5     alert(1);
6
7     // alert("вложенное событие")
8     menu.dispatchEvent(new CustomEvent("menu-open", {
9       bubbles: true
10    }));
11
12    alert(2);
13  };
14
15  document.addEventListener('menu-open', () => alert('в.
16 </script>
```

Меню (нажми меня)

Порядок вывода: 1 → вложенное событие → 2.

Обратите внимание, что вложенное событие `menu-open` успевает всплыть и запустить обработчик на `document`. Обработка вложенного события полностью завершается до того, как управление возвращается во внешний код (`onclick`).

Это справедливо не только для `dispatchEvent`, но и для других ситуаций. JavaScript в обработчике события может вызвать другие методы, которые приведут к другим событиям – они тоже обрабатываются синхронно.

Если нам это не подходит, то мы можем либо поместить `dispatchEvent` (или любой другой код, инициирующий события) в конец обработчика `onclick`, либо, если это неудобно, можно обернуть генерацию события в `setTimeout` с нулевой задержкой:



```
1 <button id="menu">Меню (нажми меня)</button>
2
3 <script>
4   menu.onclick = function() {
5     alert(1);
6
7     // alert(2)
8     setTimeout(() => menu.dispatchEvent(new CustomEvent
9       bubbles: true
10    }));
11
12    alert(2);
13  };
14
15  document.addEventListener('menu-open', () => alert('в.
16 </script>
```

Теперь `dispatchEvent` запускается асинхронно после исполнения текущего кода, включая `mouse.onclick`, поэтому обработчики полностью независимы.

Новый порядок вывода: 1 → 2 → вложенное событие.

Итого

Чтобы сгенерировать событие из кода, вначале надо создать объект события.

Базовый конструктор `Event(name, options)` принимает обязательное имя события и `options` – объект с двумя свойствами:

- `bubbles: true` чтобы событие всплывало.

Раздел

[Введение в события](#)

Навигация по уроку

Конструктор Event

Метод `dispatchEvent`

Пример всплытия

MouseEvent, KeyboardEvent и другие

Пользовательские события

`event.preventDefault()`

Вложенные события
обрабатываются синхронно

Итого

Комментарии

Поделиться



Редактировать на GitHub



- `cancellable: true` если мы хотим, чтобы `event.preventDefault()` работал.

Особые конструкторы встроенных событий `MouseEvent`, `KeyboardEvent` и другие принимают специфичные для каждого конкретного типа событий свойства. Например, `clientX` для событий мыши.

Для пользовательских событий стоит применять конструктор `CustomEvent`. У него есть дополнительная опция `detail`, с помощью которой можно передавать информацию в объекте события. После чего все обработчики смогут получить к ней доступ через `event.detail`.

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

Весьма часто, когда разработчик хочет сгенерировать встроенное событие – это вызвано «кривой» архитектурой кода.

Как правило, генерация встроенных событий полезна в следующих случаях:

- 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.

Пользовательские события со своими именами часто создают для улучшения архитектуры, чтобы сообщить о том, что происходит внутри наших меню, слайдеров, каруселей и т.д.

Проводим [курсы по JavaScript и фреймворкам](#). ✕



Комментарии

перед тем как писать...

