

Раздел

[Интерфейсные события](#)

Навигация по уроку

Краткая история

Типы событий указателя

Свойства событий указателя

Мульти-тач


Событие: pointercancel

Захват указателя

Итого

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠 → Браузер: документ, события, интерфейсы](#)
[→ Интерфейсные события](#) 29-го ноября 2020

События указателя

События указателя (Pointer events) – это современный способ обработки ввода с помощью различных указывающих устройств, таких как мышь, перо/стилус, сенсорный экран и так далее.

Краткая история

Сделаем небольшой обзор, чтобы вы поняли общую картину и место событий указателя среди других типов событий.

- Давным-давно, в прошлом, существовали только события мыши

Затем получили широкое распространение сенсорные устройства, в частности телефоны и планшеты. Чтобы скрипты корректно работали, они генерировали (и до сих пор генерируют) события мыши. Например, касание сенсорного экрана генерирует событие `mousedown`. Таким образом, сенсорные устройства позволяли работать с существующими веб-страницами.

Но сенсорные устройства во многих аспектах мощнее, чем мышь. Например, они позволяют касаться экрана сразу в нескольких местах («мульти-тач»). Однако, события мыши не имеют необходимых свойств для обработки таких прикосновений.

- Поэтому появились события касания (Touch events), такие как `touchstart`, `touchend`, `touchmove`, которые имеют специфичные для касаний свойства (мы не будем здесь рассматривать их подробно, потому что события указателя ещё лучше).

Но и этих событий оказалось недостаточно, так как существует много других устройств, таких как перо, у которых есть свои особенности. Кроме того, универсальный код, который отслеживал бы и события касаний и события мыши, неудобно писать.

- Для решения этих задач был внедрён стандарт Pointer Events («События Указателя»). Он предоставляет единый набор событий для всех типов указывающих устройств.

К настоящему времени спецификация [Pointer Events Level 2](#) поддерживается всеми основными браузерами, а [Pointer Events Level 3](#) находится в разработке и почти полностью совместима с Pointer Events Level 2.

Если вы не разрабатываете под старые браузеры, такие как Internet Explorer 10, Safari 12, или более ранние версии, больше нет необходимости использовать события мыши или касаний – можно переходить сразу на события указателя.

При этом ваш код будет корректно работать и с сенсорными устройствами и с мышью. Впрочем, у событий указателя есть важные особенности, которые нужно знать, чтобы их правильно использовать, без лишних сюрпризов. Мы отметим их в этой статье.

Типы событий указателя

Схема именований событий указателя похожа на события мыши:

Событие указателя	Аналогичное событие мыши
<code>pointerdown</code>	<code>mousedown</code>
<code>pointerup</code>	<code>mouseup</code>
<code>pointermove</code>	<code>mousemove</code>
<code>pointerover</code>	<code>mouseover</code>
<code>pointerout</code>	<code>mouseout</code>
<code>pointerenter</code>	<code>mouseenter</code>

Раздел

Интерфейсные события

Навигация по уроку

Краткая история

Типы событий указателя

Свойства событий указателя

Мульти-тач

Событие: pointercancel

Захват указателя

Итого

Комментарии

Поделиться



Редактировать на GitHub



Событие указателя	Аналогичное событие мыши
pointerleave	mouseleave
pointercancel	-
gotpointercapture	-
lostpointercapture	-

Как мы видим, для каждого `mouse<события>` есть соответствующее `pointer<событие>`, которое играет аналогичную роль. Также есть 3 дополнительных события указателя, у которых нет соответствующего аналога `mouse...`, скоро мы их разберём.

📌 Замена `mouse<событий>` на `pointer<события>` в коде

Мы можем заменить события `mouse...` на аналогичные `pointer...` в коде и быть уверенными, что с мышью по-прежнему всё будет работать нормально.

При этом поддержка сенсорных устройств «волшебным» образом улучшится. Хотя, возможно, кое-где понадобится добавить `touch-action: none` в CSS. Подробнее мы разберём это ниже, в секции про `pointercancel`.

Свойства событий указателя

События указателя содержат те же свойства, что и события мыши, например `clientX/Y`, `target` и т.п., и несколько дополнительных:

- `pointerId` – уникальный идентификатор указателя, вызвавшего событие.

Идентификатор генерируется браузером. Это свойство позволяет обрабатывать несколько указателей, например сенсорный экран со стилусом и мульти-тач (увидим примеры ниже).

- `pointerType` – тип указывающего устройства. Должен быть строкой с одним из значений: «mouse», «pen» или «touch».

Мы можем использовать это свойство, чтобы определять разное поведение для разных типов указателей.

- `isPrimary` – равно `true` для основного указателя (первый палец в мульти-тач).

Некоторые устройства измеряют область контакта и степень надавливания, например пальца на сенсорном экране, для этого есть дополнительные свойства:

- `width` – ширина области соприкосновения указателя (например, пальца) с устройством. Если не поддерживается, например мышью, то всегда равно 1.
- `height` – высота области соприкосновения указателя с устройством. Если не поддерживается, например мышью, то всегда равно 1.
- `pressure` – степень давления указателя в диапазоне от 0 до 1. Для устройств, которые не поддерживают давление, принимает значение 0.5 (нажато) либо 0.
- `tangentialPressure` – нормализованное тангенциальное давление.
- `tiltX`, `tiltY`, `twist` – специфичные для пера свойства, описывающие положение пера относительно сенсорной поверхности.

Эти свойства большинством устройств не поддерживаются, поэтому редко используются. При необходимости, подробности о них можно найти в [спецификации](#).

Мульти-тач

Одной из функций, которую абсолютно не поддерживают события мыши, является мульти-тач: возможность касаться сразу нескольких мест на телефоне или планшете или выполнять специальные жесты.

Раздел

[Интерфейсные события](#)

Навигация по уроку

Краткая история

Типы событий указателя

Свойства событий указателя

Мульти-тач

Событие: `pointercancel`

Захват указателя

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



События указателя позволяют обрабатывать мульти-тач с помощью свойств `pointerId` и `isPrimary`.

Вот что происходит, когда пользователь касается сенсорного экрана в одном месте, а затем в другом:

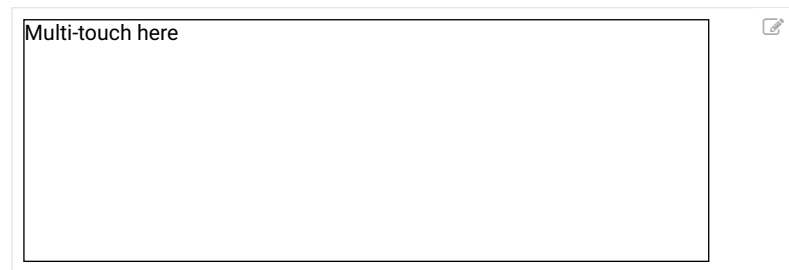
1. При касании первым пальцем:
 - происходит событие `pointerdown` со свойством `isPrimary=true` и некоторым `pointerId`.
2. При касании вторым и последующими пальцами (при остающемся первом):
 - происходит событие `pointerdown` со свойством `isPrimary=false` и уникальным `pointerId` для каждого касания.

Обратите внимание: `pointerId` присваивается не на всё устройство, а для каждого касающегося пальца. Если коснуться экрана 5 пальцами одновременно, получим 5 событий `pointerdown`, каждое со своими координатами и индивидуальным `pointerId`.

События, связанные с первым пальцем, всегда содержат свойство `isPrimary=true`.

Мы можем отслеживать несколько касающихся экрана пальцев, используя их `pointerId`. Когда пользователь перемещает, а затем убирает палец, получаем события `pointermove` и `pointerup` с тем же `pointerId`, что и при событии `pointerdown`.

Вот небольшое демо, выводящее события `pointerdown` и `pointerup`:



Обратите внимание: чтобы увидеть разницу в `pointerId/isPrimary`, вам нужно использовать устройство с сенсорным экраном, такое как телефон или планшет. Для устройств без поддержки мульти-тач, таких как мышь, всегда будет один и тот же `pointerId` со свойством `isPrimary=true`, для всех событий указателя.

Событие: `pointercancel`

Событие `pointercancel` происходит, когда текущее действие с указателем по какой-то причине прерывается, и события указателя больше не генерируются.

К таким причинам можно отнести:

- Указывающее устройство было физически выключено.
- Изменилась ориентация устройства (перевернули планшет).
- Браузер решил сам обработать действие, считая его жестом мыши, масштабированием и т.п.

Мы продемонстрируем `pointercancel` на практическом примере, чтобы увидеть, как это влияет на нас.

Допустим, мы реализуем перетаскивание («drag-and-drop») для нашего мяча, как в начале статьи [Drag'n'Drop с событиями мыши](#).

Вот последовательность действий пользователя и соответствующие события:

1. Пользователь нажимает на изображении, чтобы начать перетаскивание
 - происходит событие `pointerdown`
2. Затем он перемещает указатель, двигая изображение
 - происходит событие `pointermove` (возможно, несколько раз)
3. И тут происходит сюрприз! Браузер имеет встроенную поддержку «Drag'n'Drop» для изображений, которая запускает и перехватывает процесс перетаскивания, генерируя при этом событие `pointercancel`.

Раздел

[Интерфейсные события](#)

Навигация по уроку

Краткая история

Типы событий указателя

Свойства событий указателя

Мульти-тач

Событие: `pointercancel`

Захват указателя

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



- Теперь браузер сам обрабатывает перетаскивание изображения. У пользователя есть возможность перетащить изображение мяча даже за пределы браузера, в свою почтовую программу или файловый менеджер.
- Событий `pointermove` для нас больше не генерируется.

Таким образом, браузер «перехватывает» действие: в начале переноса `drag-and-drop` запускается событие `pointercancel`, и после этого события `pointermove` больше не генерируются.

Вот демо `drag'n'drop` с записью событий указателя (только `up/down`, `move` и `cancel`) в `textarea`:



Мы бы хотели реализовать перетаскивание самостоятельно, поэтому давайте скажем браузеру не перехватывать его.

Предотвращайте действие браузера по умолчанию, чтобы избежать `pointercancel`.

Нужно сделать две вещи:

1. Предотвратить запуск встроенного `drag'n'drop`
 - Мы можем сделать это, задав `ball.ondragstart = () => false`, как описано в статье [Drag'n'Drop с событиями мыши](#).
 - Это работает для событий мыши.
2. Для устройств с сенсорным экраном существуют другие действия браузера, связанные с касаниями, кроме `drag'n'drop`. Чтобы с ними не возникало проблем:
 - Мы можем предотвратить их, добавив в CSS свойство `#ball { touch-action: none }`.
 - Затем наш код начнёт корректно работать на устройствах с сенсорным экраном

После того, как мы это сделаем, события будут работать как и ожидается, браузер не будет перехватывать процесс и не будет вызывать событие `pointercancel`.

В данном демо произведены нужные действия:



Как вы можете видеть, событие `pointercancel` больше не срабатывает.

Теперь мы можем добавить код для перемещения мяча и наш `drag'n'drop` будет работать и для мыши и для устройств с сенсорным экраном.

Захват указателя

Захват указателя – особая возможность событий указателя.

Общая идея очень проста, но поначалу может показаться странной, так как для других событий ничего подобного просто нет.

Основной метод:

Раздел

[Интерфейсные события](#)

Навигация по уроку

Краткая история

Типы событий указателя

Свойства событий указателя

Мульти-тач

Событие: `pointercancel`

Захват указателя

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



- `elem.setPointerCapture(pointerId)` – привязывает события с данным `pointerId` к `elem`. После такого вызова все события указателя с таким `pointerId` будут иметь `elem` в качестве целевого элемента (как будто произошли над `elem`), вне зависимости от того, где в документе они произошли.

Другими словами, `elem.setPointerCapture(pointerId)` меняет `target` всех дальнейших событий с данным `pointerId` на `elem`.

Эта привязка отменяется:

- автоматически, при возникновении события `pointerup` или `pointercancel`,
- автоматически, если `elem` удаляется из документа,
- при вызове `elem.releasePointerCapture(pointerId)`.

Захват указателя используется для упрощения операций с переносом (drag'n'drop) элементов.

В качестве примера давайте вспомним реализацию слайдера из статьи [Drag'n'Drop с событиями мыши](#).

Мы делаем элемент для слайдера – полосу с «ползунком» (`thumb`) внутри.

Затем он работает так:

1. Пользователь сначала нажимает на ползунок – срабатывает `pointerdown`.
2. Затем двигает указателем его – срабатывает `pointermove`, и мы передвигаем ползунок вместе с ним.
 - ...Причём, по мере движения, указатель может покидать ползунок – перемещаться выше или ниже. При этом ползунок должен передвигаться строго по горизонтали, на одной линии с указателем.

Так что для полного отслеживания перемещения указателя, включая ниже и выше ползунка, нам пришлось поставить обработчик `pointermove` на весь документ `document`.

Такое решение выглядит слегка «грязным». Одна из проблем – это то, что движения указателя по документу могут вызвать сторонние эффекты, заставить сработать другие обработчики, не имеющие отношения к слайдеру.

Захват указателя позволяет привязать `pointermove` к `thumb` и избежать любых подобных проблем:

- Мы можем вызывать `thumb.setPointerCapture(event.pointerId)` в обработчике `pointerdown`,
- Тогда дальнейшие события указателя до `pointerup/cancel` будут привязаны к `thumb`.
- Затем, когда произойдёт `pointerup` (передвижение завершено), привязка будет автоматически удалена, нам об этом не нужно беспокоиться.

Так что, даже если пользователь будет двигать указателем по всему документу, обработчики событий будут вызваны на `thumb`. Причём все свойства объекта события, такие как `clientX/clientY`, будут корректны – захват указателя влияет только на `target/currentTarget`.

Вот основной код:

```
1 thumb.onpointerdown = function(event) {
2   // все события указателя перенаправить на thumb (пока
3   thumb.setPointerCapture(event.pointerId);
4 };
5
6 thumb.onpointermove = function(event) {
7   // перемещение ползунка: все события перенаправлены н
8   let newLeft = event.clientX - slider.getBoudingClie
9   thumb.style.left = newLeft + 'px';
10 };
11
12
```

```
13 // примечание: нет необходимости вызывать thumb.release
// при срабатывании события 'pointerup' это происходит .
```

Раздел

[Интерфейсные события](#)

Навигация по уроку

Краткая история

Типы событий указателя

Свойства событий указателя

Мульти-тач

Событие: `pointercancel`

Захват указателя

Итого

Комментарии

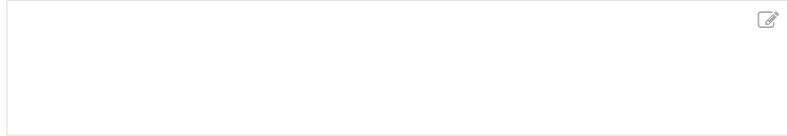
Поделиться



[Редактировать на GitHub](#)



Полное демо:



Таким образом, мы имеем два бонуса:

1. Код становится чище, поскольку нам больше не нужно добавлять/удалять обработчики для всего документа. Удаление привязки происходит автоматически.
2. Если в документе есть какие-то другие обработчики `pointermove`, то они не будут нечаянно вызваны, пока пользователь находится в процессе перетаскивания слайдера.

События при захвате указателя

Существует два связанных с захватом события:

- `gotpointercapture` срабатывает, когда элемент использует `setPointerCapture` для включения захвата.
- `lostpointercapture` срабатывает при освобождении от захвата: явно с помощью `releasePointerCapture` или автоматически, когда происходит событие `pointerup` / `pointercancel`.

Итого

События указателя позволяют одновременно обрабатывать действия с помощью мыши, касания и пера, в едином фрагменте кода.

События указателя расширяют события мыши. Мы можем заменить `mouse` на `pointer` в названиях событий и код продолжит работать для мыши, при этом получив лучшую поддержку других типов устройств.

При обработке переносов и сложных касаний, которые браузер может попытаться обработать сам, не забывайте отменять действие браузера и ставить `touch-events: none` в CSS для элементов, с которыми мы взаимодействуем.

Дополнительные возможности событий указателя:

- Поддержка мультитач с помощью `pointerId` и `isPrimary`.
- Особые свойства для определённых устройств, такие как `pressure`, `width/height` и другие.
- Захват указателя: мы можем перенаправить все события указателя на определённый элемент до наступления события `pointerup` / `pointercancel`.

На данный момент события указателя поддерживаются в основных браузерах, поэтому мы можем безопасно переходить на них, особенно если нет необходимости в поддержке IE10 и Safari 12. И даже для этих браузеров есть полифилы, которые добавляют эту поддержку.

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

Комментарии

перед тем как писать...