18-го августа 2019

Мышь: клики, кнопка, координаты

Материал на этой странице устарел, поэтому скрыт из оглавления сайта.

Более новая информация по этой теме находится на странице https://learn.javascript.ru/mouse-events-basics.

В этой главе мы глубже разберёмся со списком событий мыши, рассмотрим их общие свойства, а также те события, которые связаны с кликом.

Типы событий мыши

Условно можно разделить события на два типа: «простые» и «комплексные».

Простые события

`mousedown`

Кнопка мыши нажата над элементом.

`mouseup`

Кнопка мыши отпущена над элементом.

`mouseover`

Мышь появилась над элементом.

`mouseout`

Мышь ушла с элемента.

`mousemove`

Каждое движение мыши над элементом генерирует это событие.

Комплексные события

`click`

Вызывается при клике мышью, то есть при `mousedown`, а затем `mouseup` на одном элементе

`contextmenu`

Вызывается при клике правой кнопкой мыши на элементе.

`dblclick`

Вызывается при двойном клике по элементу.

Комплексные можно составить из простых, поэтому в теории можно было бы обойтись вообще без них. Но они есть, и это хорошо, потому что с ними удобнее.

Порядок срабатывания событий

Одно действие может вызывать несколько событий.

Например, клик вызывает сначала `mousedown` при нажатии, а затем `mouseup` и `click` при отпускании кнопки.

В тех случаях, когда одно действие генерирует несколько событий, их порядок фиксирован. То есть, обработчики вызовутся в порядке `mousedown` → `mouseup` → `click`.

Кликните по кнопке ниже и вы увидите, какие при этом происходят события. Попробуйте также двойной клик.

На тест-стенде ниже все мышиные события записываются, и если между событиями проходит больше 1 секунды, то они для удобства чтения отделяются линией. Также присутствуют свойства `which/button`, по которым можно определить кнопку мыши. Мы их рассмотрим далее.

Кликни меня левой или правой кнопкой мыши Очистить

Каждое событие обрабатывается независимо.

Например, при клике события `mouseup + click` возникают одновременно, но обрабатываются последовательно. Сначала полностью завершается обработка `mouseup` , затем запускается `click` .

Получение информации о кнопке: which

При обработке событий, связанных с кликами мыши, бывает важно знать, какая кнопка нажата.

Для получения кнопки мыши в объекте `event` есть свойство `which` .

На практике оно используется редко, т.к. обычно обработчик вешается либо `onclick` – только на левую кнопку мыши, либо `oncontextmenu` – только на правую.

Возможны следующие значения:

- `event.which == 1` – левая кнопка
- `event.which == 2` – средняя кнопка
- `event.which == 3` – правая кнопка

Это свойство не поддерживается IE8-, но его можно получить способом, описанным в конце главы.

Правый клик: oncontextmenu

Это событие срабатывает при клике правой кнопкой мыши:

```
1  <div>Правый клик на этой кнопке выведет "Клик".</div>
2  <button oncontextmenu="alert('Клик!');">Правый клик сюда</button>
```

Правый клик на этой кнопке выведет "Клик".
Правый клик сюда

При клике на кнопку выше после обработчика `oncontextmenu` будет показано обычное контекстное меню, которое браузер всегда показывает при клике правой кнопкой. Это является его действием по умолчанию.

Если мы не хотим, чтобы показывалось встроенное меню, например потому что показываем своё, специфичное для нашего приложения, то можно отменить действие по умолчанию.

В примере ниже встроенное меню показано не будет:

```
1  <button oncontextmenu="alert('Клик!');return false">Правый клик сюда</button>
```

Правый клик сюда

Модификаторы shift, alt, ctrl и meta

Во всех событиях мыши присутствует информация о нажатых клавишах-модификаторах.

Соответствующие свойства:

- `shiftKey`
- `altKey`
- `ctrlKey`
- `metaKey` (для Mac)

Например, кнопка ниже сработает только на Alt+Shift+Клик:

```
<button>Alt+Shift+Кликни меня!</button>

<script>
  document.body.children[0].onclick = function(e) {
    if (!e.altKey || !e.shiftKey) return;
    alert( 'Ура!' );
  }
</script>
```

Alt+Shift+Кликни меня!

Внимание: на Mac вместо `Ctrl` используется `Cmd`

На компьютерах под управлением Windows и Linux есть специальные клавиши `Alt`, `Shift` и `Ctrl`. На Mac есть ещё одна специальная клавиша: `Cmd`, которой соответствует свойство `metaKey`.

В большинстве случаев там, где под Windows/Linux используется `Ctrl`, на Mac используется `Cmd`. Там, где пользователь Windows нажимает `Ctrl+Enter` или `Ctrl+A`, пользователь Mac нажмёт `Cmd+Enter` или `Cmd+A`, и так далее, почти всегда `Cmd` вместо `Ctrl`.

Поэтому, если мы хотим поддерживать сочетание `Ctrl`+click или другие подобные, то под Mac имеет смысл использовать `Cmd`+click. Пользователям Mac это будет гораздо комфортнее.

Более того, даже если бы мы хотели бы заставить пользователей Mac использовать именно `Ctrl`+click – это было бы затруднительно. Дело в том, что обычный клик с зажатым `Ctrl` под Mac работает как *правый клик* и генерирует событие `oncontextmenu`, а вовсе не `onclick`, как под Windows/Linux.

Решение – чтобы пользователи обоих операционных систем работали с комфортом, в паре с `ctrlKey` нужно обязательно использовать `metaKey`.

В JS-коде это означает, что для удобства пользователей Mac нужно проверять `if (event.ctrlKey || event.metaKey)`.

Координаты в окне: clientX/Y

Все мышиные события предоставляют текущие координаты курсора в двух видах: относительно окна и относительно документа.

Пара свойств `clientX/clientY` содержит координаты курсора относительно текущего окна.

При этом, например, если ваше окно размером 500x500, а мышь находится в центре, тогда и `clientX` и `clientY` будут равны 250.

Можно как угодно прокручивать страницу, но если не двигать при этом мышь, то координаты курсора `clientX/clientY` не изменятся, потому что они считаются относительно окна, а не документа.

Проведите мышью над полем ввода, чтобы увидеть `clientX/clientY`:

```
<input onmousemove="this.value = event.clientX+':'+event.clientY">
```

В той же системе координат работает и метод `elem.getBoundingClientRect()`, возвращающий координаты элемента, а также `position:fixed`.

Относительно документа: pageX/Y

Координаты курсора относительно документа находятся в свойствах `pageX/pageY`.

Так как эти координаты – относительно левого-верхнего узла документа, а не окна, то они учитывают прокрутку. Если прокрутить страницу, а мышь не трогать, то координаты курсора `pageX/pageY` изменятся на величину прокрутки, они привязаны к конкретной точке в документе.

В IE8- этих свойств нет, но можно получить их способом, описанным в конце главы.

Проведите мышью над полем ввода, чтобы увидеть `pageX/pageY` (кроме IE8-):

```
<input onmousemove="this.value = event.pageX+':'+event.pageY">
```

В той же системе координат работает `position:absolute`, если элемент позиционируется относительно документа.

> ⚠ **Устарели: `x, y, layerX, layerY`**
>
> Некоторые браузеры поддерживают свойства `event.x/y`, `event.layerX/layerY`.
>
> Эти свойства устарели, они нестандартные и не добавляют ничего к описанным выше. Использовать их не стоит.

Особенности IE8-

Двойной клик

Все браузеры, кроме IE8-, генерируют `dblclick` *в дополнение* к другим событиям.

То есть, обычно:

- `mousedown` (нажал)
- `mouseup+click` (отжал)
- `mousedown` (нажал)
- `mouseup+click+dblclick` (отжал).

IE8- на втором клике не генерирует `mousedown` и `click`.

Получается:

- `mousedown` (нажал)
- `mouseup+click` (отжал)
- (нажал второй раз, без события)
- `mouseup+dblclick` (отжал).

Поэтому отловить двойной клик в IE8-, отслеживая только `click`, нельзя, ведь при втором нажатии его нет. Нужно именно событие `dblclick`.

Свойство which/button

В старых IE8- не поддерживалось свойство `which`, а вместо него использовалось свойство `button`, которое является 3-х битным числом, в котором каждому биту соответствует кнопка мыши. Бит установлен в 1, только если соответствующая кнопка нажата.

Чтобы его расшифровать – нужна побитовая операция `&` («битовое И»):

- `!!(button & 1) == true` (1-й бит установлен), если нажата левая кнопка,
- `!!(button & 2) == true` (2-й бит установлен), если нажата правая кнопка,
- `!!(button & 4) == true` (3-й бит установлен), если нажата средняя кнопка.

Что интересно, при этом мы можем узнать, были ли две кнопки нажаты одновременно, в то время как стандартный `which` такой возможности не даёт. Так что, в некотором смысле, свойство `button` – более мощное.

Можно легко сделать функцию, которая будет ставить свойство `which` из `button`, если его нет:

```
function fixWhich(e) {
  if (!e.which && e.button) { // если which нет, но есть button... (IE8-)
    if (e.button & 1) e.which = 1; // левая кнопка
    else if (e.button & 4) e.which = 2; // средняя кнопка
    else if (e.button & 2) e.which = 3; // правая кнопка
  }
}
```

Свойства pageX/pageY

В IE до версии 9 не поддерживаются свойства `pageX/pageY`, но их можно получить, прибавив к `clientX/clientY` величину прокрутки страницы.

Более подробно о её вычислении вы можете прочитать в разделе прокрутка страницы.

Мы же здесь приведём готовый вариант, который позволяет нам получить `pageX/pageY` для старых и совсем старых IE:

```
function fixPageXY(e) {
  if (e.pageX == null && e.clientX != null) { // если нет pageX..
    var html = document.documentElement;
    var body = document.body;

    e.pageX = e.clientX + (html.scrollLeft || body && body.scrollLeft || 0);
    e.pageX -= html.clientLeft || 0;

    e.pageY = e.clientY + (html.scrollTop || body && body.scrollTop || 0);
    e.pageY -= html.clientTop || 0;
  }
}
```

Итого

События мыши имеют следующие свойства:

- Кнопка мыши: `which` (для IE8-: нужно ставить из `button`)
- Элемент, вызвавший событие: `target`
- Координаты, относительно окна: `clientX/clientY`
- Координаты, относительно документа: `pageX/pageY` (для IE8-: нужно ставить по `clientX/Y` и прокрутке)
- Если зажата спец. клавиша, то стоит соответствующее свойство: `altKey`, `ctrlKey`, `shiftKey` или `metaKey` (Mac).
- Для поддержки `Ctrl` + `click` не забываем проверить `if (e.metaKey || e.ctrlKey)`, чтобы пользователи `Mac` тоже были довольны.

Задачи

Дерево: проверка клика на заголовке

важность: 3

Есть кликабельное JavaScript-дерево UL/LI (см. задачу Раскрывающееся дерево).

```
<ul>
  <li>Млекопитающие
    <ul>
      <li>Коровы</li>
      <li>Ослы</li>
      <li>Собаки</li>
      <li>Тигры</li>
    </ul>
  </li>
</ul>
```

При клике на заголовке его список его детей скрывается-раскрывается. Выглядит это так: (кликайте на заголовки)

- Животные
  - Млекопитающие
    - Коровы
    - Ослы
    - Собаки
    - Тигры
  - Другие
    - Змеи
    - Птицы
    - Ящерицы
- Рыбы
  - Аквариумные
    - Гуппи
    - Скалярии
  - Морские

Однако, проблема в том, что скрытие-раскрытие происходит даже при клике *вне заголовка*, на пустом пространстве справа от него.

Как скрывать/раскрывать детей только при клике на заголовок?

В задаче Раскрывающееся дерево это решено так: заголовки завёрнуты в элементы `SPAN` и проверяются клики только на них. Представим на минуту, что мы не хотим оборачивать текст в `SPAN`, а хотим оставить как есть. Например, по соображениям производительности, если дерево и так очень большое, ведь оборачивание всех заголовков в `SPAN` увеличит количество DOM-узлов в 2 раза.

Решите задачу без обёртывания заголовков в `SPAN`, используя работу с координатами.

Исходный документ содержит кликабельное дерево.

P.S. Задача – скорее на сообразительность, однако подход может быть полезен в реальной жизни.

Открыть песочницу для задачи.

решение

Проводим курсы по JavaScript и фреймворкам.

Комментарии

перед тем как писать...