27th November 2019

Mouse events basics

In this chapter we'll get into more details about mouse events and their properties.

Please note: such events may come not only from "mouse devices", but are also from other devices, such as phones and tablets, where they are emulated for compatibility.

Mouse event types

We can split mouse events into two categories: "simple" and "complex"

Simple events

The most used simple events are:

`mousedown/mouseup`

Mouse button is clicked/released over an element.

`mouseover/mouseout`

Mouse pointer comes over/out from an element.

`mousemove`

Every mouse move over an element triggers that event.

`contextmenu`

Triggers when opening a context menu is attempted. In the most common case, that happens when the right mouse button is pressed. Although, there are other ways to open a context menu, e.g. using a special keyboard key, so it's not exactly the mouse event.

…There are several other event types too, we'll cover them later.

Complex events

`click`

Triggers after `mousedown` and then `mouseup` over the same element if the left mouse button was used.

`dblclick`

Triggers after a double click over an element.

Complex events are made of simple ones, so in theory we could live without them. But they exist, and that's good, because they are convenient.

Events order

An action may trigger multiple events.

For instance, a click first triggers `mousedown`, when the button is pressed, then `mouseup` and `click` when it's released.

In cases when a single action initiates multiple events, their order is fixed. That is, the handlers are called in the order `mousedown` → `mouseup` → `click`.

Click the button below and you'll see the events. Try double-click too.

On the teststand below all mouse events are logged, and if there is more than a 1 second delay between them they are separated by a horizontal ruler.

Also we can see the `which` property that allows to detect the mouse button.

Click me with the right or the left mouse button | Clear

Getting the button: which

Click-related events always have the `which` property, which allows to get the exact mouse button.

It is not used for `click` and `contextmenu` events, because the former happens only on left-click, and the latter – only on right-click.

But if we track `mousedown` and `mouseup`, then we need it, because these events trigger on any button, so `which` allows to distinguish between "right-mousedown" and "left-mousedown".

There are the three possible values:

- `event.which == 1` – the left button
- `event.which == 2` – the middle button
- `event.which == 3` – the right button

The middle button is somewhat exotic right now and is very rarely used.

Modifiers: shift, alt, ctrl and meta

All mouse events include the information about pressed modifier keys.

Event properties:

- `shiftKey` : `Shift`
- `altKey` : `Alt` (or `Opt` for Mac)
- `ctrlKey` : `Ctrl`
- `metaKey` : `Cmd` for Mac

They are `true` if the corresponding key was pressed during the event.

For instance, the button below only works on `Alt+Shift`+click:

```
<button id="button">Alt+Shift+Click on me!</button>

<script>
  button.onclick = function(event) {
    if (event.altKey && event.shiftKey) {
      alert('Hooray!');
    }
  };
</script>
```

Alt+Shift+Click on me!

Attention: on Mac it's usually `Cmd` instead of `Ctrl`

On Windows and Linux there are modifier keys `Alt`, `Shift` and `Ctrl`. On Mac there's one more: `Cmd`, corresponding to the property `metaKey`.

In most applications, when Windows/Linux uses `Ctrl`, on Mac `Cmd` is used.

That is: where a Windows user presses `Ctrl+Enter` or `Ctrl+A`, a Mac user would press `Cmd+Enter` or `Cmd+A`, and so on.

So if we want to support combinations like `Ctrl`+click, then for Mac it makes sense to use `Cmd`+click. That's more comfortable for Mac users.

Even if we'd like to force Mac users to `Ctrl`+click – that's kind of difficult. The problem is: a left-click with `Ctrl` is interpreted as a *right-click* on MacOS, and it generates the `contextmenu` event, not `click` like Windows/Linux.

So if we want users of all operating systems to feel comfortable, then together with `ctrlKey` we should check `metaKey`.

For JS-code it means that we should check `if (event.ctrlKey || event.metaKey)`.

There are also mobile devices

Keyboard combinations are good as an addition to the workflow. So that if the visitor has a keyboard – it works. And if their device doesn't have it – then there should be another way to do the same.

Coordinates: clientX/Y, pageX/Y

All mouse events have coordinates in two flavours:

1. Window-relative: `clientX` and `clientY`.
2. Document-relative: `pageX` and `pageY`.

For instance, if we have a window of the size 500x500, and the mouse is in the left-upper corner, then `clientX` and `clientY` are `0`. And if the mouse is in the center, then `clientX` and `clientY` are `250`, no matter what place in the document it is, how far the document was scrolled. They are similar to `position:fixed`.

Move the mouse over the input field to see `clientX/clientY` (the example is in the `iframe`, so coordinates are relative to that `iframe`):

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

Mouse over me

Document-relative coordinates `pageX`, `pageY` are counted from the left-upper corner of the document, not the window. You can read more about coordinates in the chapter Coordinates.

Disabling selection

Double mouse click has a side-effect that may be disturbing in some interfaces: it selects the text.

For instance, a double-click on the text below selects it in addition to our handler:

```
<span ondblclick="alert('dblclick')">Double-click me</span>
```

Double-click me

If one presses the left mouse button and, without releasing it, moves the mouse, that also makes the selection, often unwanted.

There are multiple ways to prevent the selection, that you can read in the chapter Selection and Range.

In this particular case the most reasonable way is to prevent the browser action on `mousedown`. It prevents both these selections:

```
Before...
<b ondblclick="alert('Click!')" onmousedown="return false">
  Double-click me
</b>
...After
```

Before... **Double-click me** ...After

Now the bold element is not selected on double clicks, and pressing the left button on it won't start the selection.

Please note: the text inside it is still selectable. However, the selection should start not on the text itself, but before or after it. Usually that's fine for users.

Preventing copying

If we want to disable selection to protect our page content from copy-pasting, then we can use another event: `oncopy`.

```
<div oncopy="alert('Copying forbidden!');return false">
  Dear user,
  The copying is forbidden for you.
  If you know JS or HTML, then you can get everything from the page source
</div>
```

Dear user, The copying is forbidden for you. If you know JS or HTML, then you can get everything from the page source though.

If you try to copy a piece of text in the `<div>`, that won't work, because the default action `oncopy` is prevented.

Surely the user has access to HTML-source of the page, and can take the content from there, but not everyone knows how to do it.

Summary

Mouse events have the following properties:

- Button: `which`.
- Modifier keys (`true` if pressed): `altKey`, `ctrlKey`, `shiftKey` and `metaKey` (Mac).
  - If you want to handle `Ctrl`, then don't forget Mac users, they usually use `Cmd`, so it's better to check `if (e.metaKey || e.ctrlKey)`.
- Window-relative coordinates: `clientX/clientY`.
- Document-relative coordinates: `pageX/pageY`.

The default browser action of `mousedown` is text selection, if it's not good for the interface, then it should be prevented.

In the next chapter we'll see more details about events that follow pointer movement and how to track element changes under it.

Tasks

Selectable list

importance: 5

Create a list where elements are selectable, like in file-managers.

- A click on a list element selects only that element (adds the class `.selected`), deselects all others.
- If a click is made with `Ctrl` (`Cmd` for Mac), then the selection is toggled on the element, but other elements are not modified.

The demo:

Click on a list item to select it.

- Christopher Robin
- Winnie-the-Pooh
- Tigger
- Kanga
- Rabbit. Just rabbit.

P.S. For this task we can assume that list items are text-only. No nested tags.

P.P.S. Prevent the native browser selection of the text on clicks.

Open a sandbox for the task.

solution

Previous lesson Next lesson

Share

Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)