26th November 2019

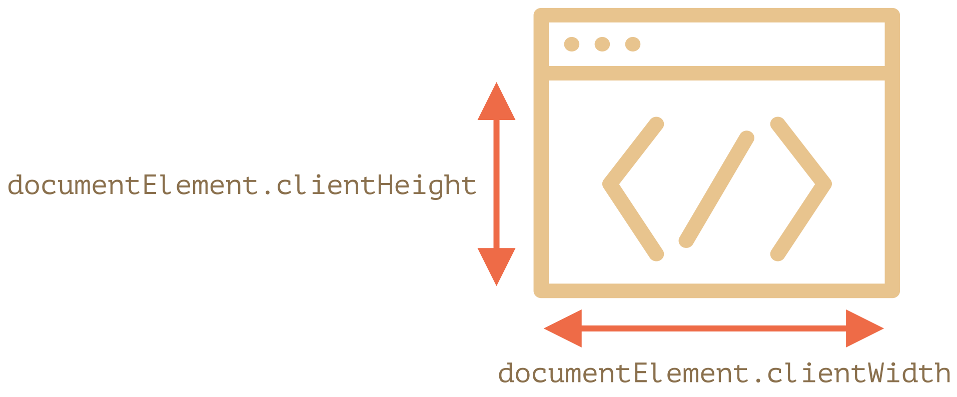
Window sizes and scrolling

How do we find the width and height of the browser window? How do we get the full width and height of the document, including the scrolled out part? How do we scroll the page using JavaScript?

For most such requests, we can use the root document element `document.documentElement` , that corresponds to the `<html>` tag. But there are additional methods and peculiarities important enough to consider.

Width/height of the window

To get window width and height we can use `clientWidth/clientHeight` of `document.documentElement` :



For instance, this button shows the height of your window:

alert(document.documentElement.clientHeight)

⚠ Not `window.innerWidth/Height`

Browsers also support properties `window.innerWidth/innerHeight` . They look like what we want. So why not to use them instead?

If there exists a scrollbar, and it occupies some space, `clientWidth/clientHeight` provide the width/height without it (subtract it). In other words, they return width/height of the visible part of the document, available for the content.

…And `window.innerWidth/innerHeight` include the scrollbar.

If there's a scrollbar, and it occupies some space, then these two lines show different values:

```
alert( window.innerWidth ); // full window width
alert( document.documentElement.clientWidth ); // window width minus the s
```

In most cases we need the *available* window width: to draw or position something. That is: inside scrollbars if there are any. So we should use `documentElement.clientHeight/Width` .

⚠ `DOCTYPE` is important

Please note: top-level geometry properties may work a little bit differently when there's no `<!DOCTYPE HTML>` in HTML. Odd things are possible.

In modern HTML we should always write `DOCTYPE` .

Width/height of the document

Theoretically, as the root document element is `document.documentElement` , and it encloses all the content, we could measure document full size as `document.documentElement.scrollWidth/scrollHeight` .

But on that element, for the whole page, these properties do not work as intended. In Chrome/Safari/Opera if there's no scroll, then `documentElement.scrollHeight` may be even less than `documentElement.clientHeight` ! Sounds like a nonsense, weird, right?

To reliably obtain the full document height, we should take the maximum of these properties:

```
let scrollHeight = Math.max(
  document.body.scrollHeight, document.documentElement.scrollHeight,
  document.body.offsetHeight, document.documentElement.offsetHeight,
  document.body.clientHeight, document.documentElement.clientHeight
);

alert('Full document height, with scrolled out part: ' + scrollHeight);
```

Why so? Better don't ask. These inconsistencies come from ancient times, not a "smart" logic.

Get the current scroll

DOM elements have their current scroll state in `elem.scrollLeft/scrollTop`.

For document scroll `document.documentElement.scrollLeft/Top` works in most browsers, except older WebKit-based ones, like Safari (bug 5991), where we should use `document.body` instead of `document.documentElement`.

Luckily, we don't have to remember these peculiarities at all, because the scroll is available in the special properties `window.pageXOffset/pageYOffset`:

```
alert('Current scroll from the top: ' + window.pageYOffset);
alert('Current scroll from the left: ' + window.pageXOffset);
```

These properties are read-only.

Scrolling: scrollTo, scrollBy, scrollIntoView

Important:

To scroll the page from JavaScript, its DOM must be fully built.

For instance, if we try to scroll the page from the script in `<head>`, it won't work.

Regular elements can be scrolled by changing `scrollTop/scrollLeft`.

We can do the same for the page using `document.documentElement.scrollTop/Left` (except Safari, where `document.body.scrollTop/Left` should be used instead).

Alternatively, there's a simpler, universal solution: special methods window.scrollBy(x,y) and window.scrollTo(pageX,pageY).

- The method `scrollBy(x,y)` scrolls the page *relative to its current position*. For instance, `scrollBy(0,10)` scrolls the page `10px` down.

  The button below demonstrates this:

  window.scrollBy(0,10)

- The method `scrollTo(pageX,pageY)` scrolls the page *to absolute coordinates*, so that the top-left corner of the visible part has coordinates `(pageX, pageY)` relative to the document's top-left corner. It's like setting `scrollLeft/scrollTop`.

  To scroll to the very beginning, we can use `scrollTo(0,0)`.

  window.scrollTo(0,0)

These methods work for all browsers the same way.

scrollIntoView

For completeness, let's cover one more method: elem.scrollIntoView(top).

The call to `elem.scrollIntoView(top)` scrolls the page to make `elem` visible. It has one argument:

- if `top=true` (that's the default), then the page will be scrolled to make `elem` appear on the top of the window. The upper edge of the element is aligned with the window top.
- if `top=false`, then the page scrolls to make `elem` appear at the bottom. The bottom edge of the element is aligned with the window bottom.

The button below scrolls the page to make itself show at the window top:

this.scrollIntoView()

And this button scrolls the page to show it at the bottom:

this.scrollIntoView(false)

Forbid the scrolling

Sometimes we need to make the document "unscrollable". For instance, when we need to cover it with a large message requiring immediate attention, and we want the visitor to interact with that message, not with the document.

To make the document unscrollable, it's enough to set `document.body.style.overflow = "hidden"`. The page will freeze on its current scroll.

Try it:

document.body.style.overflow = 'hidden'

document.body.style.overflow = ''

The first button freezes the scroll, the second one resumes it.

We can use the same technique to "freeze" the scroll for other elements, not just for `document.body`.

The drawback of the method is that the scrollbar disappears. If it occupied some space, then that space is now free, and the content "jumps" to fill it.

That looks a bit odd, but can be worked around if we compare `clientWidth` before and after the freeze, and if it increased (the scrollbar disappeared) then add `padding` to `document.body` in place of the scrollbar, to keep the content width the same.

Summary

Geometry:

- Width/height of the visible part of the document (content area width/height): `document.documentElement.clientWidth/Height`
- Width/height of the whole document, with the scrolled out part:

```
let scrollHeight = Math.max(
  document.body.scrollHeight, document.documentElement.scrollHeight,
  document.body.offsetHeight, document.documentElement.offsetHeight,
  document.body.clientHeight, document.documentElement.clientHeight
);
```

Scrolling:

- Read the current scroll: `window.pageYOffset/pageXOffset` .
- Change the current scroll:
  - `window.scrollTo(pageX,pageY)` – absolute coordinates,
  - `window.scrollBy(x,y)` – scroll relative the current place,
  - `elem.scrollIntoView(top)` – scroll to make `elem` visible (align with the top/bottom of the window).

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…)