nth-child vs nth-of-type

Sep 06, 2016 css

The `nth-child()` and `nth-of-type()` selectors are “structural” pseudo-classes, which are classes that allow us to select elements based on information within the document tree that cannot typically be represented by other simple selectors.

In the case of `nth-child()` and `nth-of-type()`, the extra information is the element’s position in the document tree in relation to its parent and siblings. Although these two pseudo-classes are very similar, they work in fundamentally different ways.

How `nth-child()` Works

The `nth-child()` pseudo-class is used to match an element based on a number, which represents the element’s position amongst it’s siblings. More specifically, the number represents the number of siblings that exist before the element in the document tree (minus 1).

This number is expressed as a function, `an+b`, where `n` is the index, and `a` and `b` are any integers we pass. For example, to select every element, we could write any of the following -

CSS

```css
:nth-child(1n+0) { /* styles */ }
:nth-child(n+0) { /* styles */ }
:nth-child(1n) { /* styles */ }
```

In addition to using this function, we can pass a single integer, e.g. `:nth-child(1)` or use one of the set keywords, `odd` or `even`. These keywords are alternatives for writing out the functional notation for selecting every odd or even numbered element.

CSS

```
:nth-child(odd) { /* styles for odd elements */ }
:nth-child(2n+1) { /* styles for odd elements */ }

:nth-child(even) { /* styles for even elements */ }
:nth-child(2n+0) { /* styles for even elements */ }
```

When using `:nth-child()` on its own, it is simple enough to predict which element will be selected. For example, using this markup -

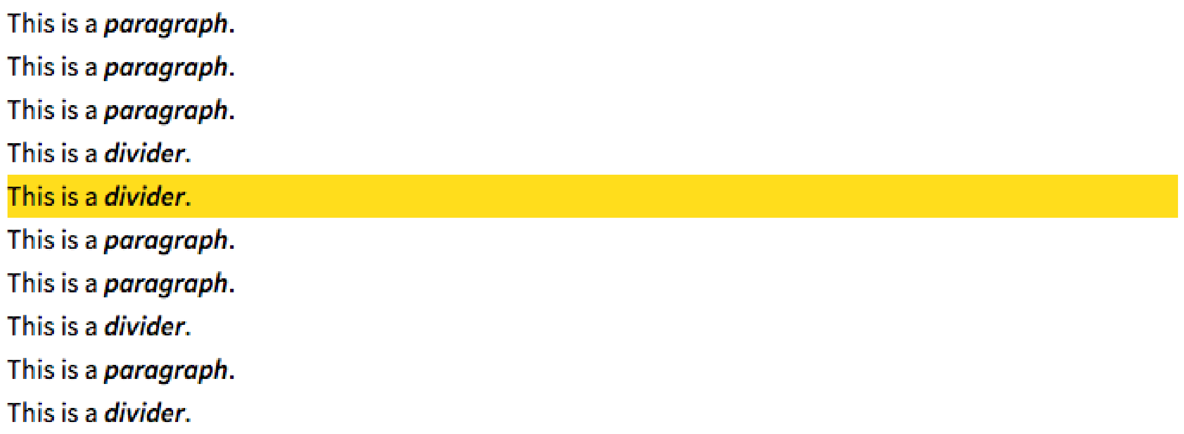
HTML

```
<div class="example">
    <p>This is a <em>paragraph</em>.</p>
    <p>This is a <em>paragraph</em>.</p>
    <p>This is a <em>paragraph</em>.</p>
    <div>This is a <em>divider</em>.</div>
    <div>This is a <em>divider</em>.</div> <!-- Element to select -->
    <p>This is a <em>paragraph</em>.</p>
    <p>This is a <em>paragraph</em>.</p>
    <div>This is a <em>divider</em>.</div>
    <p>This is a <em>paragraph</em>.</p>
    <div>This is a <em>divider</em>.</div>
</div>
```

If we wanted to select the fifth element, the div, we could simply write the following -

CSS

```
.example :nth-child(5) { background: #ffdb3a; }
```



This is a *paragraph*.
This is a *paragraph*.
This is a *paragraph*.
This is a *divider*.
This is a *divider*.
This is a *paragraph*.
This is a *paragraph*.
This is a *divider*.
This is a *paragraph*.
This is a *divider*.

However, unexpected results may occur when there are multiple types of elements, and we need to combine the `:nth-child()` pseudo-class with type or class selectors. For example, to select that same div element again, we may be tempted to write the following -

CSS

```css
.example div:nth-child(2) { background: #ffdb3a; }
```

Note - Will not work

The reason this will not work is because the element that selector is targeting does not actually exist. Using the above selector, the user agent will go through the following steps -

1. Select all the child elements of `.example`
2. Find the second element in that list, irrespective of type
3. Check if that element is a type of `div`

Since the second element in the document tree is a paragraph, not a div, nothing is selected. If we wanted to select the second div element, we would have to use the `nth-of-type()` pseudo-class.

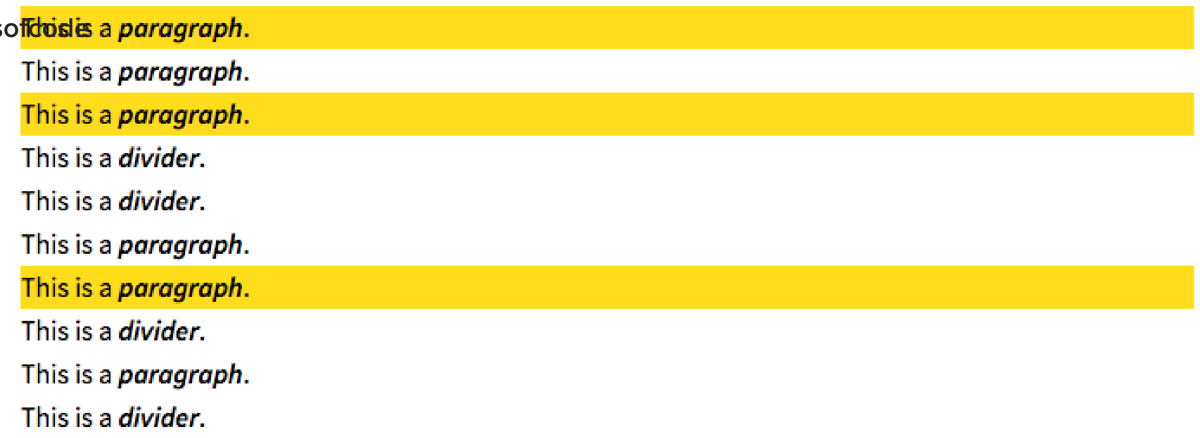
How `nth-of-type()` Works

The `nth-of-type()` pseudo-class, like `nth-child()`, is used to match an element based on a number. This number, however, represents the element's position within *only those of its siblings that are of the same element type*.

This number can also be expressed as a function, or using the keywords `even` or `odd`. Using the example markup above, we can select all the odd paragraphs by writing -

CSS

```css
.example p:nth-of-type(odd) { background: #ffdb3a; }
```


This is a *paragraph*.
This is a *paragraph*.
This is a *paragraph*.
This is a *divider*.
This is a *divider*.
This is a *paragraph*.
This is a *paragraph*.
This is a *divider*.
This is a *paragraph*.
This is a *divider*.

When we use this selector, the user agent goes through the following steps -

1. Select all the child elements of `.example` that are of the type `p`
2. Create a new list of only these elements
3. Select the odd numbers from that list

Because of this, we can now select the second div, the fifth child of `.example` -

```css
.example div:nth-of-type(2) { /* styles */ }
```

Other "nth" Pseudo-Classes

Besides `nth-child()` and `nth-of-type()`, there are a number of other structural pseudo-classes we can use to select elements based on their position within their siblings. Like `nth-child()` and `nth-of-type()`, they fall into two groups - the ones that are **type-independent**, like `nth-child()`, and the ones that are **type-dependent**, like `nth-of-type()`.

Type-independent	Type-dependent
nth-child()	nth-of-type()
nth-last-child()	nth-last-of-type()
first-child()	first-of-type()
last-child()	last-of-type()

Type-independent	Type-dependent
only-child()	only-of-type()

Counting From The End - `nth-last-child()` vs `nth-last-of-type()`

These pseudo-classes work like `nth-child()` and `nth-of-type()`, but they begin counting from the last element in the group of siblings instead of the first.

```css
.example :nth-last-child(1) { background: #a6cae7; }
.example p:nth-last-of-type(1) { background: #ffdb3a; }
```

The First Element - `first-child()` vs `first-of-type()`

The `first-child()` and `first-of-type()` pseudo-classes select the first element. They can be thought of as using the `nth-child()` and `nth-of-type()` pseudo-classes, but simply passing in a value of 1.

```css
.example :first-child() { /* styles */ }
.example :nth-child(1) { /* styles */ } /* same as above */

.example :first-of-type() { /* styles */ }
.example :nth-of-type(1) { /* styles */ } /* same as above */
```

The Last Element - `last-child()` vs `last-of-type()`

These are the opposite to the `first-child()` and `first-of-type()` pseudo-classes. They can be thought of as using the `nth-last-child()` and `nth-last-of-type()` pseudo-classes, but passing in a value of 1.

```css
.example :last-child() { /* styles */ }
.example :nth-last-child(1) { /* styles */ } /* same as above */
```

```css
.example :last-of-type() { /* styles */ }
.example :nth-last-of-type(1) { /* styles */ } /* same as above */
```

The Only Element - `only-child()` vs `only-of-type()`

Finally, these pseudo-classes will select an element that is the only child. For `only-child()` , the element must be literally the only child of it's parent, regardless of type. For `only-of-type()` , the element need only be the only child of its type.

CSS

```css
.example :only-child() { /* styles */ }

.example p:only-of-type() { /* styles */ }
```

Share on Twitter Share on Facebook Post to Reddit

Toast.js, a Library for Toast messages Previous

Next JavaScript Promises 102 - The 4 Promise Methods

7 Comments bitsofcode 1 Login

Recommend 4 Share Sort by Best

Join the discussion…

Вадим Самуйлов • 2 months ago

Hello! Thank you for your job. I want to draw your attention that the pseudo-classes without "nth" are used without brackets.

2 ^ | ˅ • Reply • Share ›

Sarbbottam • 2 months ago

Another great write up. Great job Ire!

1 ˄ | ˅ • Reply • Share ›

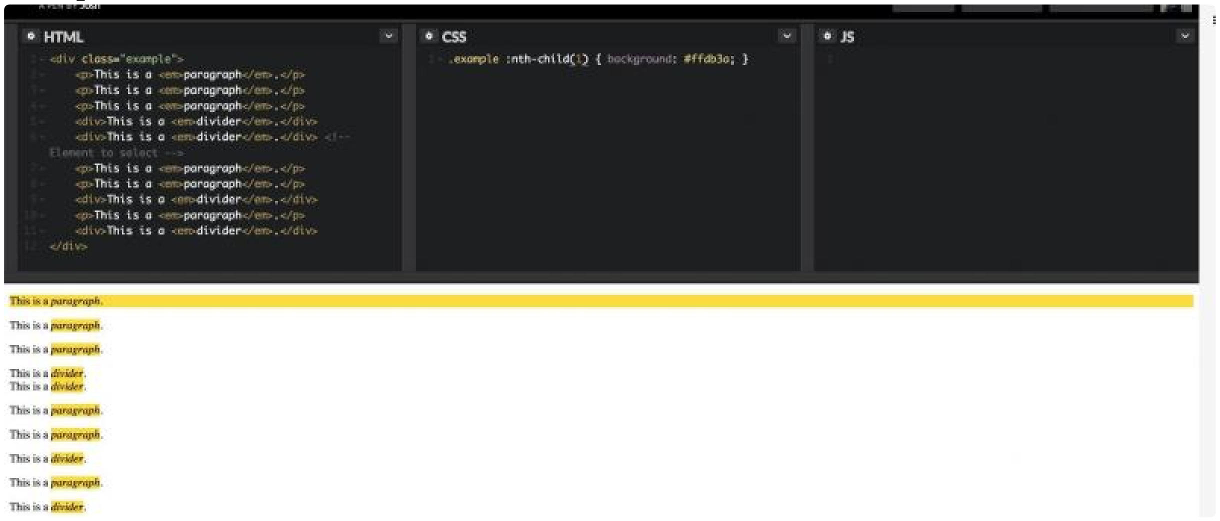
Ire Aderinokun Mod ➦ Sarbbottam • 2 months ago

Thank you Sarbbottam!

˄ | ˅ • Reply • Share ›

Joshua Ugba • 2 months ago

Succinct write up. It is important to note that when using these selectors, there's a cascading that happens which can lead to the undesired results. The image below shows an example of such.



```html
<div class="example">
    <p>This is a <em>paragraph</em>.</p>
    <p>This is a <em>paragraph</em>.</p>
    <p>This is a <em>paragraph</em>.</p>
    <div>This is a <em>divider</em>.</div>
    <div>This is a <em>divider</em>.</div> <!--
Element to select -->
    <p>This is a <em>paragraph</em>.</p>
    <p>This is a <em>paragraph</em>.</p>
    <div>This is a <em>divider</em>.</div>
    <p>This is a <em>paragraph</em>.</p>
    <div>This is a <em>divider</em>.</div>
</div>
```

```css
.example :nth-child(1) { background: #ffdb3a; }
```

```
This is a paragraph.
This is a paragraph.
This is a paragraph.
This is a divider.
This is a divider.
This is a paragraph.
This is a paragraph.
This is a divider.
This is a paragraph.
This is a divider.
```

1 ˄ | ˅ • Reply • Share ›

Ire Aderinokun Mod ➦ Joshua Ugba • 2 months ago

Thanks. Yes the reason that happens is because it is targeting EVRERY element that is the first child amongst it's siblings. Because the *elements are within the*

and <div>, they are also selected.

˄ | ˅ • Reply • Share ›

Sujin Park • a month ago

Great article! Thank you.

˄ | ˅ • Reply • Share ›

Ire Aderinokun Mod ➦ Sujin Park • a month ago

You're welcome :)

˄ | ˅ • Reply • Share ›

ALSO ON **BITSOFCODE**

A Gulp Workflow for Building HTML Email
10 comments • 7 months ago•

Making botsofcode - An Introduction to Twitter Bots

Ire Aderinokun — Hey!Yeah it can be a bit difficult to get your head around at first. Basically, all the work happens in the `src`

1 comment • 3 months ago•

Adam Recvlohe — A good read! I will definitely get this setup soon.

Using Jekyll for Rapid CSS Testing

3 comments • 5 months ago•

Emmanuel Yusufu — Thank you very much.

The :target Trick

66 comments • 4 months ago•

viki53 — You can also use `#0` to close the target: this won't cause any jump (zero cannot be a valid id, so the browser won't try to