Clear

Theory

28 minutes reading

Some CSS properties work together. It means that when you use one, you will probably need to use the other property, e.g. when you use the `width` property, you will probably use `height` as well.

The properties `float` and `clear` are no different. When we use the `float` property, we will probably need to use `clear`. You may be wondering why we need to use `clear` and what it is for. Don't worry, that's what we are going to find out in this topic.

1 required topic
Float
1 dependent topic
Introduction to Flexbox

§1. What is 'clear' for?

We will use the `clear` property together with `float` because they complement each other. When we apply `float` to an element, all the elements that are inside the same container will float too. That's why we use `clear`: to "clear" the float and make the page flow back to its default state starting from the element on which we used `clear`.

Now let's see how we can use `clear`, and what values we can use along with it.

- `none`: The element will float normally according to the `float` of the previous element. This is the default value.
- `left`: The element won't float if one of the previous elements contains the value `float: left`.
- `right`: The element won't float if one of the previous elements contains the value `float: right`.
- `both`: The element won't float if the previous element contains one of the floating values, `left` or `right`.

§2. Normal floating

As you've learned in the previous topic, changing the `float` property of an element affects the next element that is inside the same container. Let's take a look at a simple example of how this happens.

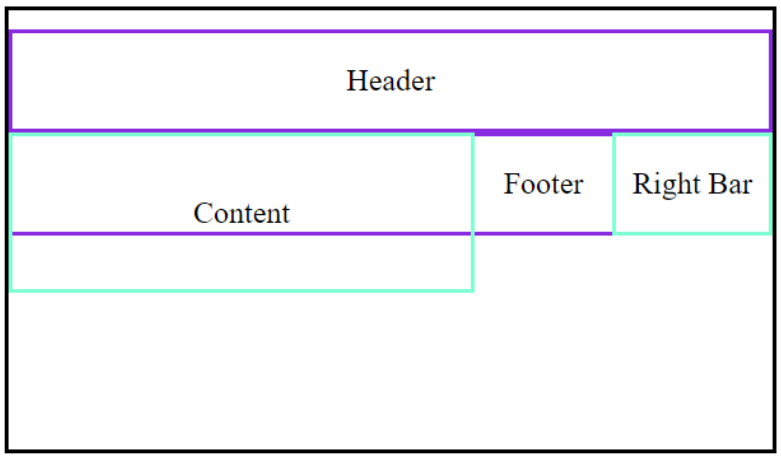
```
<div class="main-container">
  <div class="header">Header</div>
  <div class="right-bar">Right Bar</div>
  <div class="content">Content</div>
  <div class="footer">Footer</div>
</div>
```

```
.main-container {
  border: 2px solid black;
  height: 200px;
  text-align: center;
  width: 400px;
}

.header, .footer {
  border: inherit;
  border-color: blueviolet;
  height: 50px;
  line-height: 50px;
  width: 100%;
}

.content {
  border: inherit;
  border-color: aqua;
  float: left;
  height: 80px;
  line-height: 80px;
  width: 60%;
}

.right-bar {
  border: inherit;
  border-color: aqua;
  float: right;
  height: 50px;
  line-height: 50px;
  width: 20%;
}
```

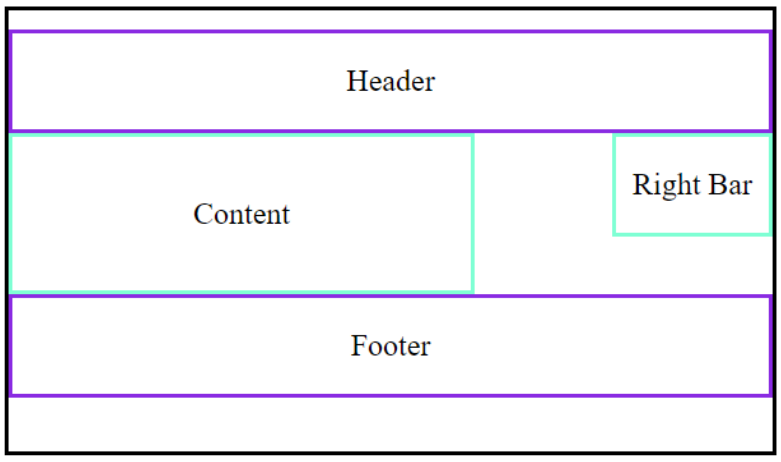


The element *footer* is now floating between the other two elements. As you may remember, `float` changes the flow of the page and since the two elements are floating, *footer* is pushed in the middle between the other two elements. But what if we want *footer* to be below the other two elements? This is where `clear` comes in handy.

§3. Using clear

The most common value used with `clear` is `both`. Let's apply it to our *footer* element and see what happens.

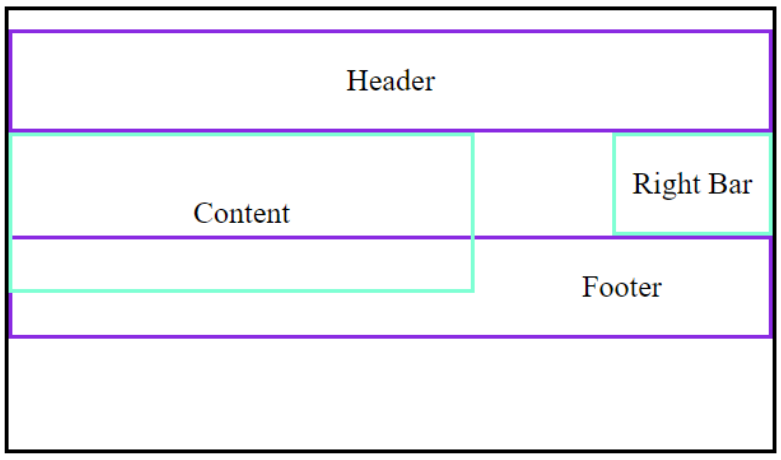
```
.footer {
  clear: both;
}
```



And voila! Like magic, our *footer* is no longer in the middle. Now any element we create after the *footer* element will no longer float because the float has been cleared.

With `clear: left` and `clear: right`, the behavior is a little different. To get a better idea let's apply `clear: right` to our *footer*.

```
.footer {
  clear: right;
}
```

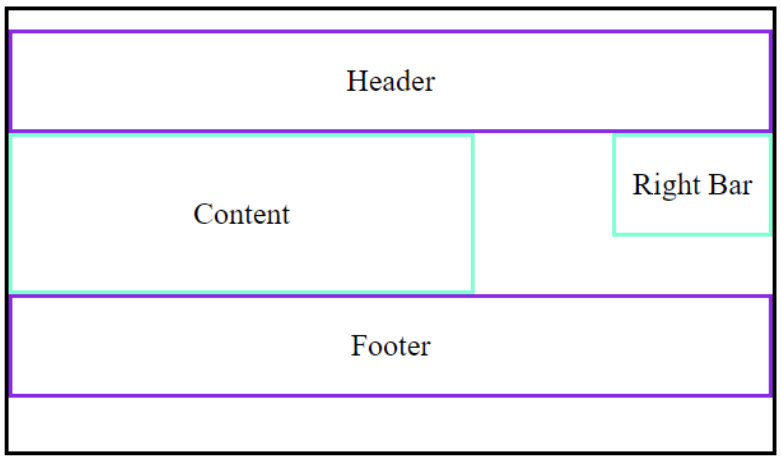


Strange, right? This happened because our *right bar* is smaller than the *content* element, and since both elements float and we cleared only the right float, the *footer* will be positioned below the *right bar*.

We also notice that our *right bar* element comes before the *content* element, but as the float changes, the flow of the page is no longer respected.

Let's see how it will behave with `clear: left`:

```
.footer {
  clear: left;
}
```



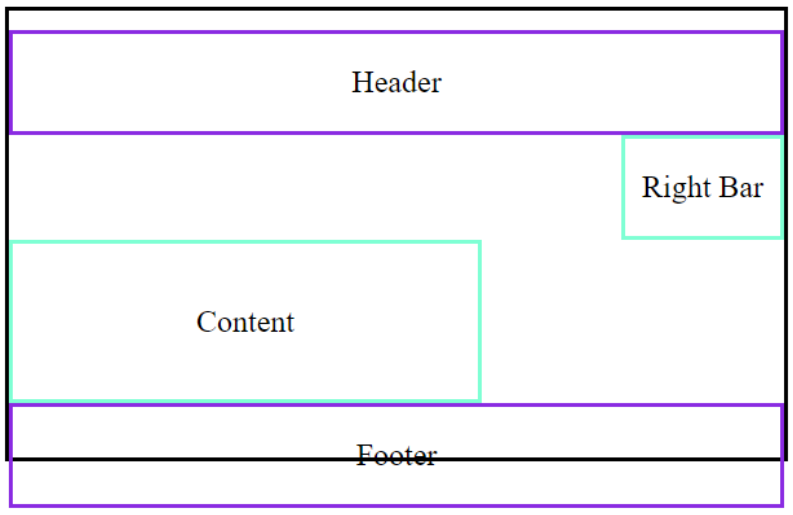
We get the same result as `clear: both`, since *content* is larger than the *right bar* element. With the `float: left` value, *footer* will be positioned after it.

§4. Unexpected result

When working with `float` and `clear` we have to be careful because sometimes the elements will not behave the way we expect them to. Let's review some examples.

If we remove `clear` from *footer* and apply it to the *content* element, this is what will happen:

```
.content {
  clear: both;
}
```

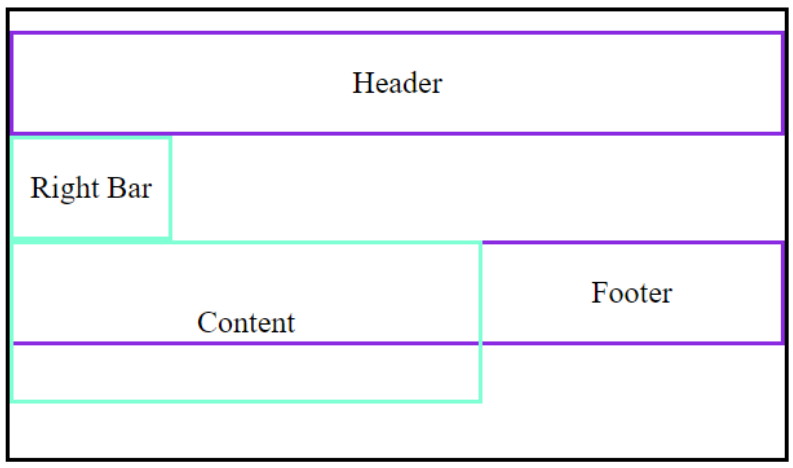


As you can see, the *footer* element has been pushed out of the main div. This happened because our div has a fixed height of `200px` and there's not enough space to fit all the elements.

Let's see one more example, now our *right bar* with `clear` and our *content* with `float: left`.

```
.right-bar {
  clear: both;
}

.content {
  float: left;
}
```



Since we removed `clear` from our *footer,* it will be positioned next to the *content* that is floating on the left.

§5. Conclusion

We've learned that when we combine the two properties, `float` and `clear`, we can construct our page layout in different ways. With the `clear` property we can return our elements to the normal flow of the web page and get better control over the positioning of the elements, thus preventing unwanted results.

The behavior of the elements with `float` and `clear` may not turn out the way we want. That's why for the page layout it is recommended to use other CSS properties, like `flexbox` and `grid`.

Now that you have learned a little more about how to combine these two properties, how about some exercises?

Report a typo

Thanks for your feedback!

Start practicing

Verify to skip

Table of contents: