

CSS Specificity Explained

You said:



explain CSS specifcuty in detail hwo to calculatw with examples

ChatGPT said:



CSS specificity determines which styles are applied to an element when multiple conflicting rules are present. The specificity of a CSS rule is calculated based on the types of selectors used. The higher the specificity, the more precedence the rule has.

Specificity Calculation

Specificity is calculated as a score, with different types of selectors contributing to different parts of the score. The specificity score is typically represented as a four-part value: `(a, b, c, d)`, where:

- **a:** Inline styles (`style=""` in HTML) have the highest specificity.
- **b:** IDs (`#id`) are next in specificity.
- **c:** Classes (`.class`), attributes (`[type="text"]`), and pseudo-classes (`:hover`, `:nth-child`) have a lower specificity.
- **d:** Element selectors (e.g., `div`, `p`) and pseudo-elements (`::before`, `::after`) have the lowest specificity.

When comparing specificity, the highest value in each segment is considered first. For example, `(1, 0, 0, 0)` is more specific than `(0, 10, 0, 0)`.

Specificity Example

Let's look at some examples to clarify:

CSS

```
/* Example 1 */
div p {
  color: blue; /* (0, 0, 0, 2) */
}

#main p {
  color: red; /* (0, 1, 0, 1) */
}

<style>
  p {
    color: green; /* Inline styles (1, 0, 0, 0) */
  }
</style>
```

1. `div p`:`
 - Contains two element selectors: `div`` and `p``.
 - Specificity: `(0, 0, 0, 2)``.
2. `#main p`:`
 - Contains an ID selector: `#main`` and an element selector: `p``.
 - Specificity: `(0, 1, 0, 1)``.
3. **Inline style applied to `p``:**
 - Inline style directly applied to the element.
 - Specificity: `(1, 0, 0, 0)``.

In this case, the inline style will override the others because its specificity `(1, 0, 0, 0)`` is higher than the others.

Another Example with More Selectors

CSS

```
/* Example 2 */
div { /* (0, 0, 0, 1) */
  color: black;
}

.container div { /* (0, 0, 1, 1) */
  color: gray;
}

#main .container div { /* (0, 1, 1, 1) */
  color: white;
}
```

1. `div`:`
 - Specificity: `(0, 0, 0, 1)``.
2. `.container div`:`
 - Specificity: `(0, 0, 1, 1)``.
3. `#main .container div`:`
 - Specificity: `(0, 1, 1, 1)``.

In this case, `#main .container div`` has the highest specificity, so `div`` elements within `.container`` inside an element with `id="main"` will have their color set to white.

Universal Selector and Specificity

The universal selector (`*`) and combinators (`+`, `>`, `~`, ```) do not contribute to specificity.

CSS

```
* {
  color: yellow; /* (0, 0, 0, 0) */
}

p {
  color: orange; /* (0, 0, 0, 1) */
}
```

Here, the `p` selector has a higher specificity `(0, 0, 0, 1)` than the universal selector `(0, 0, 0, 0)`, so the paragraph text will be orange, not yellow.

Important and Specificity

The `!important` declaration can override any specificity, but if multiple rules use `!important`, specificity is still considered.

```
CSS

p {
  color: blue !important; /* (0, 0, 0, 1) + !important */
}

p.special {
  color: red !important; /* (0, 0, 1, 1) + !important */
}
```

In this case, the `p.special` rule will apply because it has higher specificity, despite both using `!important`.

Summary

- **Inline styles:** `(1, 0, 0, 0)`
- **ID selectors:** `(0, 1, 0, 0)`
- **Class, attribute, pseudo-class selectors:** `(0, 0, 1, 0)`
- **Element and pseudo-element selectors:** `(0, 0, 0, 1)`

Higher specificity selectors take precedence. If specificity is the same, the later one in the CSS file wins.

You said:



give scores

ChatGPT said:



Here's how you would calculate the specificity scores for various selectors:

1. Element Selector

```
CSS

div {
  color: black;
}
```

Specificity: `(0, 0, 0, 1)`

- `a = 0` (no inline styles)
- `b = 0` (no IDs)
- `c = 0` (no classes, attributes, or pseudo-classes)
- `d = 1` (1 element selector)

2. Class Selector

CSS

```
.container {
  color: gray;
}
```

Specificity: `(0, 0, 1, 0)`

- `a = 0` (no inline styles)
- `b = 0` (no IDs)
- `c = 1` (1 class selector)
- `d = 0` (no element selectors)

3. ID Selector

CSS

```
#main {
  color: white;
}
```

Specificity: `(0, 1, 0, 0)`

- `a = 0` (no inline styles)
- `b = 1` (1 ID selector)
- `c = 0` (no classes, attributes, or pseudo-classes)
- `d = 0` (no element selectors)

4. Inline Styles

html

```
<p style="color: red;">
```

Specificity: `(1, 0, 0, 0)`

- `a = 1` (inline style)
- `b = 0` (no IDs)
- `c = 0` (no classes, attributes, or pseudo-classes)
- `d = 0` (no element selectors)

5. Combination of Selectors

CSS

```
#header .menu li {
  color: blue;
}
```

Specificity: `(0, 1, 1, 2)`

- `a = 0` (no inline styles)
- `b = 1` (1 ID selector `#header`)
- `c = 1` (1 class selector `.menu`)
- `d = 2` (2 element selectors `li`)

6. Attribute Selector

CSS

```
input[type="text"] {
  color: green;
}
```

Specificity: `(0, 0, 1, 1)`

- `a = 0` (no inline styles)
- `b = 0` (no IDs)
- `c = 1` (1 attribute selector `[type="text"]`)
- `d = 1` (1 element selector `input`)

7. Pseudo-class Selector

CSS

```
a:hover {
  color: yellow;
}
```

Specificity: `(0, 0, 1, 1)`

- `a = 0` (no inline styles)
- `b = 0` (no IDs)
- `c = 1` (1 pseudo-class `:hover`)
- `d = 1` (1 element selector `a`)

8. Universal Selector

CSS

```
* {
  color: purple;
}
```

```
}
```

Specificity: `(0, 0, 0, 0)`

- Universal selectors do not contribute to specificity.

9. `!important`

CSS

```
p {
  color: orange !important;
}
```

Specificity: `(0, 0, 0, 1)` + `!important`

- `!important` is not part of the specificity score but overrides other rules regardless of their specificity. However, if multiple `!important` rules exist, the one with the higher specificity score wins.

Summary Table:

Selector	Specificity Score
`div`	`(0, 0, 0, 1)`
`.container`	`(0, 0, 1, 0)`
`#main`	`(0, 1, 0, 0)`
Inline style	`(1, 0, 0, 0)`
`#header .menu li`	`(0, 1, 1, 2)`
`input[type="text"]`	`(0, 0, 1, 1)`
`a:hover`	`(0, 0, 1, 1)`
`*`	`(0, 0, 0, 0)`
`p { color: orange !important;}`	`(0, 0, 0, 1)` + `!important`

This specificity score helps determine which styles apply when multiple rules target the same element. The higher the score, the more likely that rule will be applied.

You said:



give more example

ChatGPT said:



Let's dive into more examples with varying levels of specificity to better understand how specificity scores are calculated and how they affect which CSS rule is applied.

1. Multiple Classes and Elements

CSS

```
.nav .item span {
  color: blue;
}
```

Specificity: `(0, 0, 2, 1)`

- `a = 0` (no inline styles)
- `b = 0` (no IDs)
- `c = 2` (two class selectors: `.nav`, `.item`)
- `d = 1` (one element selector: `span`)

2. ID and Class Combination

CSS

```
#sidebar .widget {
  color: green;
}
```

Specificity: `(0, 1, 1, 0)`

- `a = 0` (no inline styles)
- `b = 1` (one ID selector: `#sidebar`)
- `c = 1` (one class selector: `.widget`)
- `d = 0` (no element selectors)

3. Class, Attribute, and Pseudo-class

CSS

```
.card[data-type="primary"]:hover {
  color: red;
}
```

Specificity: `(0, 0, 2, 0)`

- `a = 0` (no inline styles)
- `b = 0` (no IDs)
- `c = 2` (one class selector: `.card`, one attribute selector: `[data-type="primary"]`, one pseudo-class: `:hover`)
- `d = 0` (no element selectors)

4. Element, Class, ID, and Pseudo-element

CSS

```
h1#title.subtitle::before {
  content: "Prefix";
}
```

Specificity: `(0, 1, 1, 2)`

- `a = 0` (no inline styles)
- `b = 1` (one ID selector: `#title`)
- `c = 1` (one class selector: `.subtitle`)
- `d = 2` (one element selector: `h1`, one pseudo-element: `::before`)

5. Complex Example with Descendant Combinators

CSS

```
body header nav ul li.active a {
  color: yellow;
}
```

Specificity: `(0, 0, 2, 5)`

- `a = 0` (no inline styles)
- `b = 0` (no IDs)
- `c = 2` (two class selectors: `.active`, and `a`)
- `d = 5` (five element selectors: `body`, `header`, `nav`, `ul`, `li`)

6. ID and Inline Style Combination

CSS

```
<div id="container" style="color: purple;">
```

Specificity: `(1, 1, 0, 0)`

- `a = 1` (inline style)
- `b = 1` (one ID selector: `#container`)
- `c = 0` (no classes)
- `d = 0` (no element selectors)

7. Pseudo-class and Attribute Selector

CSS

```
input:checked[type="radio"] {
  border-color: blue;
}
```

Specificity: `(0, 0, 2, 1)`

- `a = 0` (no inline styles)
- `b = 0` (no IDs)
- `c = 2` (one pseudo-class: `:checked`, one attribute selector: `[type="radio"]`)
- `d = 1` (one element selector: `input`)

8. Multiple IDs and Classes

CSS

```
#header #logo .icon.active {
  background-color: orange;
}
```

Specificity: `(0, 2, 2, 0)`

- `a = 0` (no inline styles)
- `b = 2` (two ID selectors: `#header`, `#logo`)
- `c = 2` (two class selectors: `.icon`, `.active`)
- `d = 0` (no element selectors)

9. Universal Selector with Classes

CSS

```
*.btn:hover {
  border: 1px solid black;
}
```

Specificity: `(0, 0, 1, 0)`

- `a = 0` (no inline styles)
- `b = 0` (no IDs)
- `c = 1` (one class selector: `.btn`, one pseudo-class: `:hover`)
- `d = 0` (universal selector `*` does not add to specificity)

10. More Specific Pseudo-classes

CSS

```
a:visited {
  color: gray;
}
```

Specificity: `(0, 0, 1, 1)`

- `a = 0` (no inline styles)
- `b = 0` (no IDs)
- `c = 1` (one pseudo-class: `:visited`)

- `d = 1` (one element selector: `a`)

Summary Table for Additional Examples:

Selector	Specificity Score
<code>.nav .item span</code>	<code>(0, 0, 2, 1)</code>
<code>#sidebar .widget</code>	<code>(0, 1, 1, 0)</code>
<code>.card[data-type="primary"]:hover</code>	<code>(0, 0, 2, 0)</code>
<code>h1#title.subtitle::before</code>	<code>(0, 1, 1, 2)</code>
<code>body header nav ul li.active a</code>	<code>(0, 0, 2, 5)</code>
<code>#container (with inline style)</code>	<code>(1, 1, 0, 0)</code>
<code>input:checked[type="radio"]</code>	<code>(0, 0, 2, 1)</code>
<code>#header #logo .icon.active</code>	<code>(0, 2, 2, 0)</code>
<code>*.btn:hover</code>	<code>(0, 0, 1, 0)</code>
<code>a:visited</code>	<code>(0, 0, 1, 1)</code>

Conclusion

In CSS, the specificity score is critical for determining which styles apply when multiple rules target the same element. Understanding how to calculate specificity allows you to write more predictable and maintainable CSS.

ChatGPT can make mistakes. Check important info.