La cascada

Curso de Fundamentos

Introducción

En la lección anterior, analizamos la sintaxis básica de CSS y los selectores. Ahora es el momento de combinar nuestro conocimiento de los selectores con el lenguaje *C* de CSS: la cascada.

Resumen de la lección

Esta sección contiene una descripción general de los temas que aprenderá en esta lección.

- Qué hace la cascada.
- Especificidad y combinación de selectores CSS.
- Cómo afecta la herencia a determinadas propiedades.

La cascada de CSS

A veces, podemos tener reglas que entran en conflicto entre sí y terminamos con resultados inesperados. “Pero quería que *estos* párrafos fueran azules, ¿por qué son rojos como estos otros párrafos?” Por más frustrante que esto pueda ser, es importante entender que CSS no solo *hace* cosas en contra de nuestros deseos. CSS solo hace lo que le decimos que haga. Una excepción a esto son los estilos predeterminados que proporciona un navegador. Estos estilos predeterminados varían de un navegador a otro y son la razón por la que algunos elementos crean un gran

“espacio” entre ellos y otros elementos, o por la que los botones se ven como se ven, a pesar de que no escribimos ninguna regla CSS para darles ese estilo.

Entonces, si terminas con un comportamiento inesperado como este, es debido a estos estilos predeterminados, a no entender cómo funciona una propiedad o a no entender esta pequeña cosa llamada cascada.

La cascada es lo que determina qué reglas se aplican realmente a nuestro HTML. Hay diferentes factores que la cascada utiliza para determinar esto. Analizaremos tres de estos factores, que esperamos te ayuden a evitar esos frustrantes momentos de “Odio CSS”.

Especificidad

Una declaración CSS más específica tendrá prioridad sobre las menos específicas. Los estilos en línea, que analizamos en la lección anterior, tienen la mayor especificidad en comparación con los selectores, mientras que cada tipo de selector tiene su propio nivel de especificidad que contribuye a la especificidad de una declaración. Otros selectores contribuyen a la especificidad, pero nos centraremos solo en los que hemos analizado hasta ahora:

1. Selectores de ID (selector más específico)
2. Selectores de clase
3. Selectores de tipo

La especificidad solo se tendrá en cuenta cuando un elemento tenga múltiples declaraciones conflictivas dirigidas a él, como si fuera un desempate. Un selector de ID siempre superará a cualquier número de selectores de clase, un selector de clase siempre superará a cualquier número de selectores de tipo y un selector de tipo siempre superará a cualquier número de selectores menos específicos. Cuando no hay una declaración con un selector de mayor especificidad, una regla con un mayor número de selectores del mismo tipo tendrá prioridad sobre otra regla con menos selectores del mismo tipo.

Veamos algunos ejemplos rápidos para visualizar cómo funciona la especificidad. Considere el siguiente código HTML y CSS:

```
<!-- index.html -->

<div class="main">
  <div class="list subsection">Red text</div>
</div>
```

```
/* rule 1 */
.subsection {
  color: blue;
}

/* rule 2 */
.main .list {
  color: red;
}
```

En el ejemplo anterior, ambas reglas utilizan solo selectores de clase, pero la regla 2 es más específica porque utiliza más selectores de clase, por lo que la `color: red` declaración tendría prioridad.

Ahora, cambiemos las cosas un poco:

```
<!-- index.html -->

<div class="main">
  <div class="list" id="subsection">Blue text</div>
</div>
```

```
/* rule 1 */
#subsection {
  color: blue;
}

/* rule 2 */
```

```
.main .list {
  color: red;
}
```

In the example above, despite rule 2 having more class selectors than ID selectors, rule 1 is more specific because ID beats class. In this case, the `color: blue` declaration would take precedence.

And a bit more complex:

```
<!-- index.html -->

<div class="main">
  <div class="list" id="subsection">Red text on yellow backgro
</div>
```

```
/* rule 1 */
#subsection {
  background-color: yellow;
  color: blue;
}

/* rule 2 */
.main #subsection {
 color: red;
}
```

In this final example, the first rule uses an ID selector, while the second rule combines an ID selector with a class selector. Therefore, neither rule is using a more specific selector than the other. The cascade then checks the number of each selector type. Both rules have only one ID selector, but rule 2 has a class selector in addition to the ID selector, so rule 2 has a higher specificity!

While the `color: red` declaration would take precedence, the `background-color: yellow` declaration would still be applied since there's no conflicting declaration for it.

> Not everything adds to specificity
>
> When comparing selectors, you may come across special symbols for the universal selector (`*`) as well as combinators (`+` , `~` , `>` , and an empty space). These symbols do not add any specificity in and of themselves.

```
/* rule 1 */
.class.second-class {
  font-size: 12px;
}

/* rule 2 */
.class .second-class {
  font-size: 24px;
}
```

Here both rule 1 and rule 2 have the same specificity. Rule 1 uses a chaining selector (no space) and rule 2 uses a descendant combinator (the empty space). But both rules have two classes and the combinator symbol itself does not add to the specificity.

```
/* rule 1 */
.class.second-class {
  font-size: 12px;
}

/* rule 2 */
.class > .second-class {
  font-size: 24px;
}
```

This example shows the same thing. Even though rule 2 is using a child combinator (`>`), this does not change the specificity value. Both rules still have two classes so they have the same specificity values.

```
/* rule 1 */
* {
  color: black;
}

/* rule 2 */
h1 {
  color: orange;
}
```

In this example, rule 2 would have higher specificity and the `orange` value would take precedence for this element. Rule 2 uses a type selector, which has the lowest specificity value. But rule 1 uses the universal selector (`*`) which has no specificity value.

Inheritance

Inheritance refers to certain CSS properties that, when applied to an element, are inherited by that element's descendants, even if we don't explicitly write a rule for those descendants. Typography-based properties (`color`, `font-size`, `font-family`, etc.) are usually inherited, while most other properties aren't.

The exception to this is when directly targeting an element, as this always beats inheritance:

```
<!-- index.html -->

<div id="parent">
  <div class="child"></div>
</div>
```

```
/* styles.css */

#parent {
  color: red;
}
```

```
7
8   .child {
9     color: blue;
    }
```

Despite the `parent` element having a higher specificity with an ID, the `child` element would have the `color: blue` style applied since that declaration directly targets it, while `color: red` from the parent is only inherited.

Rule order

The final factor, the end of the line, the tie-breaker of the tie-breakers. Let's say that after every other factor has been taken into account, there are still multiple conflicting rules targeting an element. How does the cascade determine which rule to apply?

Whichever rule was the *last* defined is the winner.

```
1   /* styles.css */
2
3   .alert {
4     color: red;
5   }
6
7   .warning {
8     color: yellow;
9   }
```

Para un elemento que tiene las clases `alert` y `warning` , la cascada se ejecutaría a través de todos los demás factores, incluida la herencia (ninguna aquí) y la especificidad (ninguna regla es más específica que la otra). Dado que la `.warning` regla fue la última definida y ningún otro factor pudo determinar qué regla aplicar, es la que se aplica al elemento.

Asignación

1. Completa el ejercicio en el directorio de nuestro repositorio de ejercicios CSS `foundations/cascade` (recuerda que las instrucciones están en el README):

   - `01-cascade-fix`

   Nota: Las soluciones de estos ejercicios se pueden encontrar en la `solution` carpeta de cada ejercicio.

2. ¿Recuerdas la página de recetas que creaste como práctica en la sección Fundamentos de HTML? Bueno, parece bastante *simple* , ¿no? ¡Vamos a solucionarlo agregándole algo de CSS!

   - La forma en que lo diseñes es completamente libre, pero debes usar el método CSS externo (para esta práctica y en adelante). También debes intentar usar varias de las propiedades mencionadas en la lección anterior (color, color de fondo, propiedades de tipografía, etc.). Tómate un tiempo para jugar con las distintas propiedades para tener una idea de lo que hacen. Por ahora, no te preocupes en absoluto por hacer que se vea *bien* . Esto es solo para practicar y acostumbrarte a escribir CSS, no para hacer algo para presumir en tu currículum.
   - Aún no hemos explicado cómo usar una fuente personalizada para la `font-family` propiedad, así que por ahora echa un vistazo a Fuentes CSS para ver una lista de familias de fuentes genéricas que puedes usar, y a Fuentes CSS Web Safe para ver una lista de fuentes que son seguras para la web. Seguras para la web significa que son fuentes que están instaladas en prácticamente todas las computadoras o dispositivos (pero asegúrate de incluir una familia de fuentes genéricas como respaldo).

Comprobación de conocimientos

Las siguientes preguntas son una oportunidad para reflexionar sobre temas clave de esta lección. Si no puede responder una pregunta, haga clic en ella para revisar el material, pero tenga en cuenta que no se espera que memorice o domine este conocimiento.

- Entre una regla que utiliza un selector de clase y una regla que utiliza tres selectores de tipo, ¿qué regla tiene la mayor especificidad?

Recursos adicionales

Esta sección contiene enlaces útiles a contenido relacionado. No es obligatoria, por lo que se la puede considerar complementaria.

- CSS Cascade es una lectura excelente e interactiva que analiza con un poco más de detalle otros factores que afectan qué reglas CSS realmente se terminan aplicando.
- En CSS Specificity Explained, Kevin Powell analiza varios ejemplos de especificidad y brinda algunos consejos para evitar luchar con la especificidad.
- La calculadora de especificidad CSS le permite completar sus propios selectores y calcular y visualizar su especificidad.
- La Referencia de propiedades CSS de Mozilla se puede utilizar para saber si una propiedad CSS en particular se hereda o no; busque el campo **Heredado dentro de la sección Definición formal** . Aquí hay un ejemplo de la propiedad CSS `color` .
- Scrim interactivo en CSS Cascade.

Mejorar en GitHub Informar un problema Ver registro de cambios de la lección

Ver curso

Marcar como completado

Próxima lección

¡Apóyanos!

El Proyecto Odin está financiado por la comunidad. ¡Únase a nosotros para ayudar a estudiantes de todo el mundo apoyando el Proyecto Odin!

Más información

Dona ahora

THE ODIN PROJECT

Educación en codificación de alta calidad mantenida por una comunidad de código abierto.

Sobre nosotros

Acerca de

Equipo

Blog

Casos de éxito

Apoyo

Preguntas frecuentes

Contribuir

Contáctenos

Guías

Guías de la comunidad

Guías de instalación

Legal

Términos

Privacidad