Unidad 7: SASS

Extend y Mixins en SASS

¡Despedite de la repetición en CSS con extends y mixins de SASS!

¿Cansado de escribir el mismo código CSS una y otra vez para diferentes elementos? ¿Te gustaría que tus hojas de estilo sean más organizadas y fáciles de mantener?

¡La solución está en SASS!

Este preprocesador de CSS te ofrece dos herramientas poderosas para reutilizar estilos: **@extend** y **@mixin**.

SASS es un preprocesador de CSS que permite escribir estilos más mantenibles y reutilizables. Dos características clave de SASS son `@extend` y `@mixin`, que facilitan la reutilización de reglas de estilo.

@extend

El `@extend` en SASS permite compartir un conjunto de reglas de estilo entre diferentes selectores. Esto ayuda a evitar la duplicación de código y a mantener los estilos más limpios.

Ejemplo de Uso de @extend

/ Definición de una clase base

```
.button {
  padding: 10px 20px;
  border-radius: 5px;
  color: white;
  background-color: blue;
```

```
}

// Extensión de la clase base en otras clases

.primary-button {

  @extend .button;

  background-color: blue;

}

.secondary-button {

  @extend .button;

  background-color: gray;

}
```

En este ejemplo, `.primary-button` y `.secondary-button` heredan las propiedades de `.button`, pero pueden agregar o modificar propiedades específicas.

@mixin

Los mixins en SASS permiten definir estilos que se pueden reutilizar en diferentes lugares de tu CSS. A diferencia de `@extend`, los mixins pueden aceptar parámetros, lo que los hace muy flexibles y poderosos.

Ejemplo de Uso de @mixin

```
// Definición de un mixin con parámetros

@mixin button-styles($padding, $border-radius, $bg-color) {

  padding: $padding;

  border-radius: $border-radius;

  color: white;

  background-color: $bg-color;

}

// Uso del mixin en clases
```

```
.primary-button {

  @include button-styles(10px 20px, 5px, blue);

}

.secondary-button {

  @include button-styles(10px 20px, 5px, gray);

}
```

En este ejemplo, el mixin `button-styles` se utiliza para aplicar estilos de botón a `.primary-button` y `.secondary-button` con diferentes colores de fondo.

Cuándo Usar @extend y Cuándo Usar @mixin

- **@extend:** Úsalo cuando tengas un conjunto de reglas de estilo que necesites compartir entre varios selectores sin necesidad de parametrización, es decir, heredar estilos comunes sin necesidad de parámetros
- **@mixin:** Úsalo cuando necesites reutilizar reglas de estilo que requieran parámetros o cuando necesites incluir un bloque de estilos en múltiples lugares.

Conclusión

El uso de `@extend` y `@mixin` en SASS puede mejorar significativamente la mantenibilidad y reutilización de tu CSS. Con estas herramientas, puedes escribir estilos más DRY (Don't Repeat Yourself) y gestionar tus hojas de estilo de manera más eficiente.

Condicionales y Bucles en SASS

¿Te encontraste en una situación donde los estilos CSS necesitan cambiar dependiendo de ciertas condiciones o necesitas generar código repetitivo para diferentes elementos?

SASS, además de permitir escribir CSS más mantenible y eficiente, incluye características de programación como condicionales y bucles. Estas características hacen que SASS sea aún más poderoso y flexible para gestionar estilos dinámicos.

Condicionales en SASS

Los condicionales en SASS permiten aplicar estilos en función de condiciones específicas. Se utilizan con la directiva `@if`.

Ejemplo de Uso de Condicionales

scssCopiar código

```
$theme: light;

.button {
  @if $theme == light {
    background-color: white;
    color: black;
  } @else if $theme == dark {
    background-color: black;
    color: white;
  } @else {
    background-color: gray;
    color: white;
  }
}
```

En este ejemplo, los estilos de `.button` cambian según el valor de la variable `$theme`.

Bucles en SASS

Los bucles en SASS permiten iterar sobre listas, mapas o rangos de números, aplicando estilos de manera repetitiva y dinámica. Se utilizan con las directivas `@for`, `@each`, y `@while`.

Ejemplo de Uso de @for

scssCopiar código

```
@for $i from 1 through 3 {
  .column-#{$i} {
    width: 100% / $i;
```

```
  }
}
```

Este bucle genera clases `.column-1`, `.column-2`, y `.column-3` con anchos correspondientes.

Ejemplo de Uso de @each

scssCopiar código

```
$colors: red, green, blue;

@each $color in $colors {
  .text-#{$color} {
    color: $color;
  }
}
```

Este bucle genera clases `.text-red`, `.text-green`, y `.text-blue` con colores correspondientes.

Ejemplo de Uso de @while

scssCopiar código

```
$i: 6;

@while $i > 0 {
  .item-#{$i} {
    font-size: 2em * $i;
  }
  $i: $i - 1;
}
```

Este bucle genera clases `.item-6`, `.item-5`, hasta `.item-1` con tamaños de fuente decrecientes.

Combinando Condicionales y Bucles

Puedes combinar condicionales y bucles para crear estilos más complejos y dinámicos.

Ejemplo Combinado

scssCopiar código

```
$themes: (light, dark);

$sizes: (small, medium, large);

@each $theme in $themes {

  @each $size in $sizes {

    .#{$theme}-#{$size} {

      @if $theme == light {

        background-color: white;

      } @else {

        background-color: black;

      }

      @if $size == small {

        font-size: 1em;

      } @else if $size == medium {

        font-size: 1.5em;

      } @else {

        font-size: 2em;

      }

    }

  }

}
```

Este ejemplo genera clases como `.light-small`, `.dark-large`, etc., con estilos adaptados a cada combinación de tema y tamaño.

Conclusión

El uso de condicionales y bucles en SASS permite escribir hojas de estilo más eficientes, dinámicas y fáciles de mantener. Estas características de programación amplían las capacidades de CSS, facilitando la creación de diseños complejos y adaptables.

Referencias

Documentación oficial de SASS: https://sass-lang.com/