Atribuições de Responsividade, Importância e Pseudo Elementos

Todas as classes do WPF são compatíveis com o que chamamos de **atribuições**. As **atribuições** são nada mais do que comandos extras que podem ser inclusos em uma classe utilitária do WPF que impõe sobre ela comportamentos extras ou específicos. Esses comportamentos podem ser de inúmeros tipos: *responsivos, importantes ou pseudos.*

Nesta lição, veremos um pouco mais sobre cada um deles e como podemos atribuir cada uma dessas **atribuições** nas nossas classes auxiliares:

Responsividade

As **atribuições de responsividade** podem ser todas encontradas no arquivo de instruções do WPF, no arquivo `wpf.config.yaml` dentro do seu projeto. Na folha de estilos do `wpf.config.yaml` você pode encontrar uma seção chamada "**breakpoints**":

image.png

Cada um dos itens listados sobre essa lista refere-se à uma regra de estilo que define uma nova resolução em largura (utilizando pixels) para definir um ponto de quebra na renderização da sua página. Pegue '*mob: 1024*' por exemplo: Se você atribuir o prefixo **"mob:"** sob sua classe utilitária, antes do nome da classe, você estará definindo um estilo que será aplicado automaticamente em dispositivos que tenham uma **largura menor do que 1024px**. O mesmo vale para todas as outras variações definidas sob essa lista.

Por exemplo, ao utilizar a combinação de classes `flex` e `mob:grid` em um elemento `<div>`, você pode alcançar resultados interessantes:

```
<div class="dp-flex mob:dp-grid">
   <!-- Conteúdo aqui -->
</div>
```

Neste exemplo, o elemento `<div>` terá o estilo `display: flex` em contextos padrão do projeto, mas quando a largura da tela for menor que 1024px, o estilo será alterado para `display: grid`,

permitindo uma transição suave e eficiente entre layouts responsivos.

Aqui o resultado final, em um CSS padrão, ficaria assim:

```
.dp-flex{
  display: flex;
}

@media (max-width: 1024px){
  .mob\:dp-grid{
    display: grid;
  }
}
```

Se for necessário atribuir novos tipos de breakpoints para o seu projeto, você pode editar essa lista de breakpoints presentes no *wpf.config.yaml* para criar seus próprios pontos de quebra customizados para designs responsivos.

Importância

As **atribuições de importância** servem para definir, em um código CSS onde um elemento possuí duas vezes o mesmo atributo sob situações conflitantes, um ponto de partida para que o CSS descubra qual regra deve sempre se sobressair como o estilo dominante.

Leve por exemplo um elemento que precisa mudar o seu tamanho sem necessariamente estar sob um breakpoint. Ele tem um tamanho original de 50% (informado pela classe utilitária `w-50%`). Esse mesmo elemento receberá do JavaScript uma nova classe que mudará o seu tamanho para 25% (informado pela classe utilitária `w-25%`) quando o usuário clicar em algum botão dentro do site. Como são elementos conflitantes (a largura, no caso), dificilmente o navegador saberá entender que o valor de 25% é o correto a ser aplicado agora. Informando o código que o novo estilo **é mais importante** (atribuindo a classe utilitária com o modificador de importância `!w-25%`) agora o navegador saberá o momento em que o elemento precisa ter 25% de largura e o momento em que ele precisa ter 50%.

```
<div x-data="{ isNarrow: false }">
  <div class="w-50%" x-bind:class="{ '!w-25%': isNarrow }">
     <!-- Conteúdo aqui -->
  </div>
</div>
```

O resultado final disso, em um arquivo CSS comum, seria bem simples:

```
.w-50\%{
  width: 50%;
}

.\!w-25\%{
  width: 25%!important;
}
```

É possível adicionar esse tipo de comportamento à elementos que estão com **atribuições de responsividade**. Veja um exemplo:

```
<div x-data="{ isNarrow: false }">
  <div class="mob:w-50%" x-bind:class="{ 'mob:!w-25%': isNarrow }">
     <!-- Conteúdo aqui -->
  </div>
</div>
```

Pseudo Elementos

Por último, existem as atribuições que chamamos normalmente no CSS como **pseudo elementos**. Isso implica a todos os elementos virtuais que podem ser gerados pelo *DOM* via CSS, como **:active, ::placeholder, ::after, ::before** etc. No WPF, alguns desses pseudo elementos possuem um suporte nativo pela ferramenta, e podem receber estilos e classes diretamente informando o pseudo-elemento que você deseja ativar/inserir para aquele elemento HTML.

Para definir um pseudo elemento à um elemento HTML é muito simples: basta você escrevê-lo antes da classe e de qualquer outra atribuição adicional que esteja inserindo à uma classe utilitária WPF. Daremos um exemplo:

```
<div class="bg(c-bg-1) hover:bg(c-bg-2)">
   <!-- Conteúdo aqui -->
</div>
```

No exemplo acima, o elemento possuí a cor de background `var(--c-bg-1)` que é automaticamente alterada para `var(--c-bg-2)` no momento em que o usuário passar o cursor do mouse por cima do elemento, ou seja, através da ativação do pseudo elemento **:hover**. Esses tipos de atribuições não geram um resultado diretamente no CSS, mas são lidos de maneira dinâmica diretamente no navegador.

É importante lembrar que as **atribuições de pseudo elementos funcionam em conjunto com os outros tipos de atribuições, como as de responsividade ou importância**. Veja um exemplo:

```
<div class="bg(c-bg-1) mob:bg(c-bg-2) hover:mob:bg(c-bg-3)">
    <!-- Conteúdo aqui -->
</div>
```

No exemplo acima, a versão natural do componente não possuí um comportamento com o pseudo elemento **:hover**. Esse comportamento foi instruído para acontecer somente em um contexto responsivo, com duas outras cores completamente diferentes.

Revisão #3
Criado 5 novembro 2024 13:35:37 por Lyautey Maluf Neto
Atualizado 11 julho 2025 20:34:58 por Lyautey Maluf Neto