Dominando a Manipulação do DOM com `getElementsByClassName`

1. INTRODUÇÃO AO `GETELEMENTSBYCLASSNAME`

Formalmente, **`document.getElementsByClassName('nome-da-classe')`** é um método do objeto `document` que varre o DOM em busca de todos os elementos que possuem a classe especificada em seu atributo `class`. Sua principal função é retornar uma coleção desses elementos, permitindo-nos aplicar alterações, adicionar eventos ou extrair informações de forma agrupada.

O ponto mais crucial a se entender sobre este método é o que ele retorna: uma **`HTMLCollection`**. À primeira vista, uma `HTMLCollection` pode parecer um *array* — ela possui uma propriedade `length` e permite o acesso a elementos por um índice numérico. No entanto, é fundamental destacar que **ela não é um array**. Essa distinção é vital, pois a `HTMLCollection` não possui os métodos de iteração modernos e poderosos disponíveis para *arrays*, como `.map()` ou `.forEach()`. Com essa base estabelecida, vamos preparar nosso ambiente de prática para ver como aplicar esse seletor de forma eficaz e como contornar suas limitações.

1.1. Preparando o Ambiente de Prática (HTML e CSS)

Para testar e visualizar de forma clara a manipulação de elementos via JavaScript, é fundamental ter uma estrutura HTML bem definida e um estilo CSS correspondente. A seguir, apresentamos o código base para nossos exemplos. Conforme a recomendação do instrutor, digitar o código em vez de simplesmente copiar e colar é uma excelente forma de praticar e fixar o conhecimento. Primeiro, vamos criar a estrutura de nossa página com alguns elementos `div` que servirão como nossos alvos.

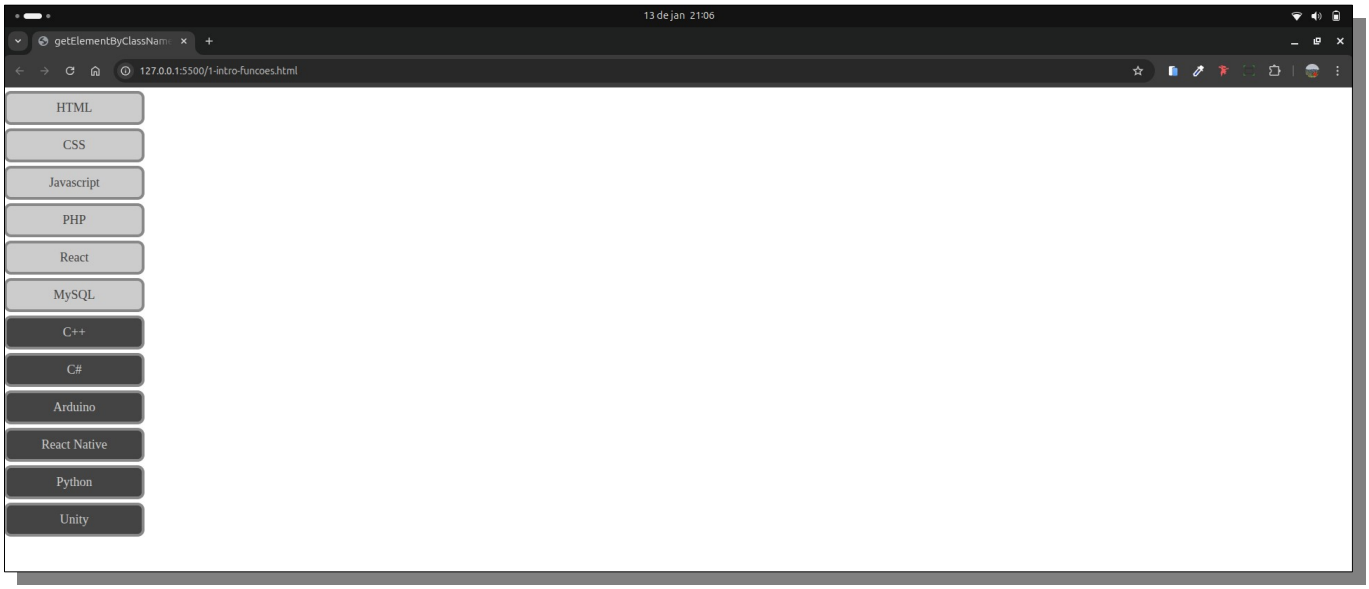
```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Aula 32 - getElementsByClassName</title>
    <link rel="stylesheet" type="text/css" href="estilos.css"/>
</head>
<body>
    <main>
        <div id="c1" class="curso todos C1">HTML</div>
        <div id="c2" class="curso todos C1">CSS</div>
        <div id="c3" class="curso todos C1">Javascript</div>
        <div id="c4" class="curso todos C1">PHP</div>
        <div id="c5" class="curso todos C1">React</div>
        <div id="c6" class="curso todos C1">MySQL</div>
        <div id="c7" class="curso todos C2">C++</div>
        <div id="c8" class="curso todos C2">C#</div>
        <div id="c9" class="curso todos C2">Arduino</div>
        <div id="c10" class="curso todos C2">React Native</div>
        <div id="c11" class="curso todos C2">Python</div>
        <div id="c12" class="curso todos C2">Unity</div>
    </main>
    <script src="script.js"></script>
</body>
</html>
```

Nesta estrutura, observe que cada `div` possui múltiplas classes. Todos os 12 elementos compartilham a classe `curso` e `todos`, enquanto os seis primeiros também possuem a classe `C1` e os seis últimos possuem a `C2`. Agora, vamos adicionar o estilo para dar vida a esses elementos.

```
* {
    padding: 0px;
    margin: 0px;
    border: none;
    box-sizing: border-box;
    font-size: large;
}

.curso {
    display: flex;
    justify-content: center;
    align-items: center;
    width: 200px;
    border: 4px solid #888;
    border-radius: 10px;
    padding: 10px;
    margin: 5px 0px;
    cursor: pointer;
}

.curso:hover {
    border-color: #f00;
}

.C1 {
    background-color: #ccc;
    color: #444;
}

.C2 {
    background-color: #444;
    color: #ccc;
}
```

O CSS acima define um estilo base para todos os elementos com a classe `.curso`, utilizando Flexbox para centralizar o conteúdo. As classes `.C1` e `.C2` criam variações visuais, alterando as cores de fundo e da fonte. Além disso, um efeito `hover` foi adicionado para mudar a cor da borda para vermelho quando o cursor do mouse passa sobre um elemento. O resultado é uma coluna vertical de 12 caixas. Graças às classes `.C1` e `.C2`, os seis primeiros cursos terão um fundo cinza-claro com texto escuro, enquanto os seis últimos terão um fundo cinza-escuro com texto claro, criando dois grupos visuais distintos. Com a

estrutura visual pronta, estamos preparados para começar a selecionar e manipular esses elementos usando JavaScript.



1.2. Selecionando Elementos com `getElementsByClassName`

O primeiro passo para interagir com qualquer elemento na página é selecioná-lo. Com o `getElementsByClassName`, podemos capturar todos os elementos que compartilham uma classe com uma única linha de código. A sintaxe básica é `document.getElementsByClassName('nome-da-classe')`. Vamos começar selecionando todos os elementos que possuem a classe `curso`.

```
const cursosTodos = document.getElementsByClassName('curso');
console.log(cursosTodos);
```

Ao executar este código e inspecionar o console do navegador, veremos o resultado: uma `HTMLCollection` contendo os 12 `divs` da nossa página. A propriedade `length` indicará o valor `12`, e podemos expandir o objeto no console para inspecionar cada elemento individualmente.

1.2.1. A Diferença Crucial: `HTMLCollection` vs. Array

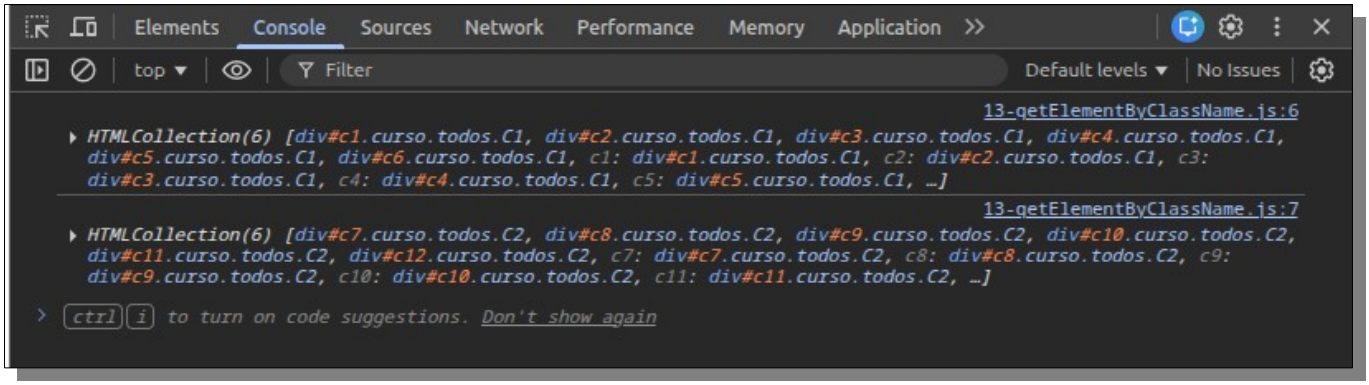
Como mencionado, o `HTMLCollection` retornado não é um *array* verdadeiro. Isso significa que não podemos usar diretamente métodos de iteração modernos e convenientes, como `.map()`, `.filter()` ou `.forEach()`. Tentar fazer isso resultaria em um erro. Felizmente, a solução para isso é simples e elegante. Para resolver isso, vamos modificar nossa declaração para converter essa coleção em um *Array* utilizando o **operador *Spread* `(...)`**.

```
const cursosTodos = [...document.getElementsByClassName('curso')];
console.log(cursosTodos);
```

Ao envolver a chamada do método com colchetes e prefixá-la com `...`, estamos "espalhando" cada item da `HTMLCollection` dentro de um novo *array*. Se você executar este novo código, o `console.log` agora exibirá um `Array` nativo, e não mais uma `HTMLCollection`. Essa transformação é poderosa, pois "desbloqueia" todo o arsenal de métodos de *array* do JavaScript, permitindo-nos manipular os elementos do DOM de forma muito mais eficiente e declarativa. Podemos também criar seleções mais específicas. Vamos criar constantes separadas para os cursos das classes `C1` e `C2`:

```
const cursosC1 = [...document.getElementsByClassName('C1')];
const cursosC2 = [...document.getElementsByClassName('C2')];

console.log(cursosC1);
console.log(cursosC2);
```

O console agora mostrará dois *arrays* distintos, cada um com 6 elementos, correspondendo aos grupos `C1` e `C2`. Com essas coleções em mãos, podemos começar a aplicar mudanças dinâmicas na página.



1.3. Aplicação Prática: Adicionando Classes Dinamicamente

Um dos usos mais comuns da manipulação do DOM é modificar a aparência dos elementos em resposta a uma ação ou lógica. Faremos isso adicionando uma nova classe CSS dinamicamente aos elementos que selecionamos.

1.3.1.1. Passo 1: Criar a Classe de Destaque

Primeiro, vamos adicionar uma nova classe ao nosso arquivo CSS que definirá o estilo de "destaque".

```
.destaque {
    background-color: #800 !important;
    color: #fcc !important;
    border-color: #f00 !important;
}
```

Neste trecho, a regra **`!important`** desempenha um papel fundamental. Como os nossos elementos já possuem estilos de `background-color`, `color` e `border-color` definidos pelas classes `.C1` e `.C2`, precisamos garantir que as propriedades da nova classe `.destaque` tenham prioridade. O `!important` força o navegador a aplicar esses estilos, sobrescrevendo quaisquer regras conflitantes que não tenham essa mesma diretiva.

1.3.1.2. Passo 2: Iterar e Adicionar a Classe

Agora, vamos usar o método `.map()` (que só é possível porque convertemos nossa coleção para um array) para percorrer cada elemento e adicionar a classe `destaque`. Vamos construir o entendimento em etapas: Primeiro, aplicaremos a classe a **todos** os elementos:

```
const cursosTodos = [...document.getElementsByClassName('curso')];

cursosTodos.map((el) => {
    el.classList.add('destaque');
});
```

O método `el.classList.add('destaque')` é a forma moderna e segura de adicionar uma classe a um elemento sem interferir nas classes já existentes. O resultado visual é que todos os 12 cursos recebem o estilo de destaque. Agora, vamos refinar nossa seleção para aplicar o destaque apenas aos cursos do grupo `C1`:

```
const cursosC1 = [...document.getElementsByClassName('C1')];

cursosC1.map((el) => {
    el.classList.add('destaque');
});
```

Com essa alteração, apenas os seis primeiros elementos mudam de aparência. Finalmente, como nosso exemplo final, vamos aplicar o destaque apenas ao grupo `C2`:

```
const cursosC2 = [...document.getElementsByClassName('C2')];

cursosC2.map((el) => {
    el.classList.add('destaque');
});
```

O resultado visual agora é que somente os seis últimos elementos (aqueles com a classe `C2`) terão sua aparência alterada para o estilo de destaque vermelho. Como exercício, modifique a constante de

`cursosC2` para `cursosC1` e observe como o destaque é aplicado ao outro grupo de elementos. Essa prática é a melhor forma de solidificar o conceito. Essa técnica progressiva demonstra como a combinação de `getElementsByClassName` com métodos de *array* nos permite aplicar lógica e modificações em massa de forma seletiva e poderosa.

1.4. Acessando um Elemento Específico da Coleção

Enquanto converter para um *array* é essencial para iteração com métodos como `.map()` ou `.forEach()`, existem cenários onde este passo é desnecessário. Se seu objetivo é simplesmente acessar um único e conhecido elemento pela sua posição, você pode fazer isso diretamente na `HTMLCollection` retornada por `getElementsByClassName`. Esta abordagem é ligeiramente mais performática, pois evita a criação de um novo *array* em memória. Para isso, usamos a sintaxe de colchetes `[índice]`, assim como faríamos com um *array*. Para reforçar o conceito de índice zero, vamos primeiro selecionar o primeiro elemento da coleção:

```
const primeiroCurso = document.getElementsByClassName('curso')[0];
console.log(primeiroCurso);
```

Isso retornará o primeiro `div` da nossa lista, o que corresponde ao curso de "HTML". Agora, vamos selecionar um elemento mais específico no meio da coleção:

```
const cursoEspecial = document.getElementsByClassName('curso')[6];
console.log(cursoEspecial);
```

É importante lembrar que os índices de uma `HTMLCollection` (e de *arrays*) começam em `0`. Portanto, ao solicitar o índice `[6]`, estamos na verdade selecionando o **sétimo** elemento da coleção. Em nosso exemplo, isso corresponde ao `div` com o texto "C++" (`id="c7"`), que é o primeiro item do grupo `C2`. Essa abordagem é extremamente útil quando a posição de um elemento na página é conhecida e você precisa direcioná-lo de forma rápida e direta.

2. CONCLUSÃO E PRÓXIMOS PASSOS

Nesta apostila, exploramos em detalhes o método `getElementsByClassName`, uma ferramenta fundamental para a manipulação do DOM. Os principais aprendizados foram:

- A seleção de múltiplos elementos com `getElementsByClassName` baseada em suas classes CSS.
- A natureza do retorno (`HTMLCollection`), sua diferença para um Array, e a estratégia de conversão para Array para habilitar métodos de iteração modernos como `.map()`.
- A manipulação em massa de elementos, como adicionar uma classe a um grupo inteiro, através da iteração com métodos como `.map()`.
- O acesso direto a elementos individuais da coleção utilizando um índice numérico.

A maestria na manipulação do DOM é um pilar para o desenvolvimento JavaScript moderno, e `getElementsByClassName` é mais uma ferramenta valiosa no seu arsenal. O aprendizado continua, e nos próximos passos exploraremos novos métodos de seleção e o fascinante mundo dos eventos, que nos permitirá responder às interações do usuário em tempo real. Continue praticando!