

INSTITUTO MAUÁ DE TECNOLOGIA



Fundamentos Web


Front-end

Prof. Tiago Sanches da Silva

Caminho crítico de renderização

Caminho crítico de renderização

A entrega de uma experiência rápida de web exige muito trabalho do navegador. A maior parte desse trabalho não aparece para nós, desenvolvedores web: escrevemos a marcação e uma página bonita é exibida na tela. **Mas como exatamente o navegador transforma o consumo do HTML, CSS e JavaScript em pixels renderizados na tela?**



Caminho crítico de renderização

Para conseguir renderizar a página, o navegador precisa construir as árvores do **DOM** e do **CSSOM**. Como resultado, precisamos garantir a entrega mais rápida possível do HTML e do CSS ao navegador.

Você sabe o que é o **DOM**? E o **CSSOM**?

- Document Object Model
- CSS Object Model

Mas o que eles são de fato e como são montados?

DOM

A fase final de **parser** do HTML resulta no DOM.

Para entendermos melhor o que o processamento que está por traz da geração do DOM vamos olhar uma página extremamente simples para entendermos o processo.

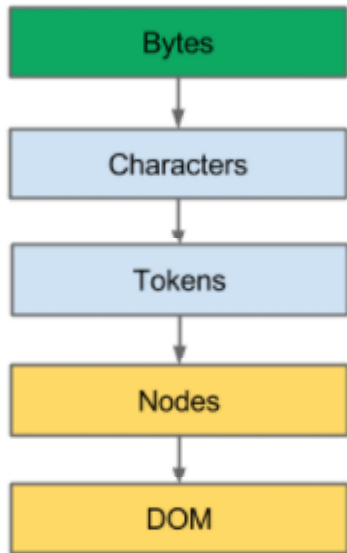
Hello students!

```
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link href="style.css" rel="stylesheet">
    <title>Critical Path</title>
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div></div>
  </body>
</html>
```

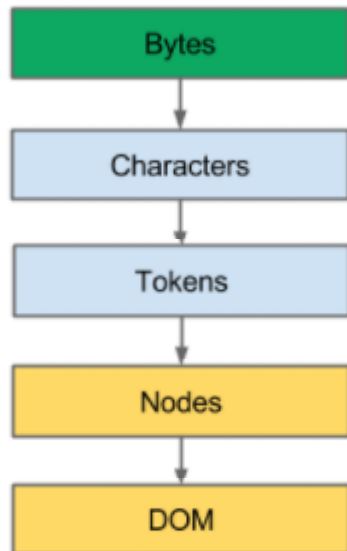
Hello students!

You're
Awesome!

DOM

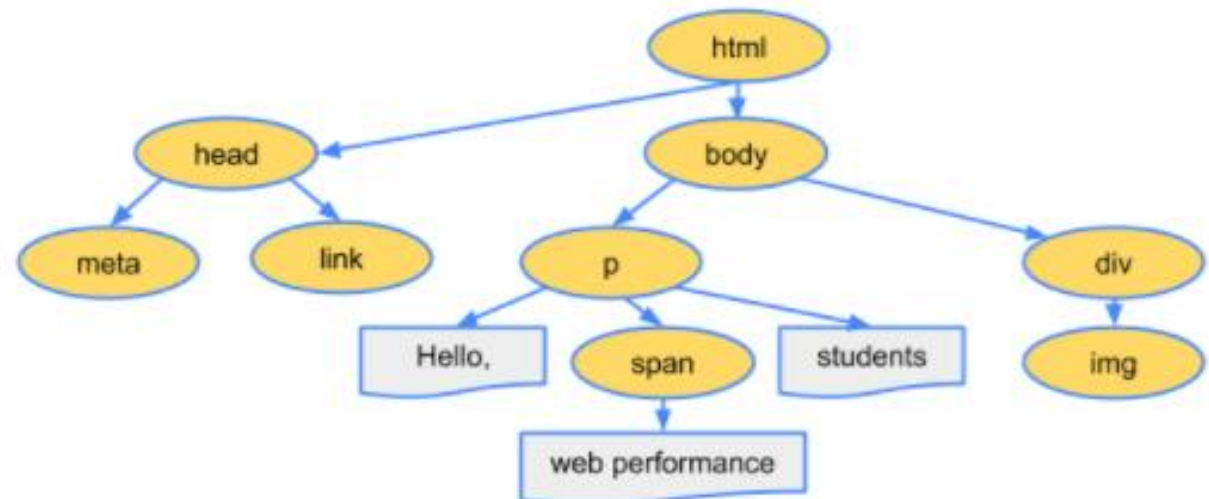
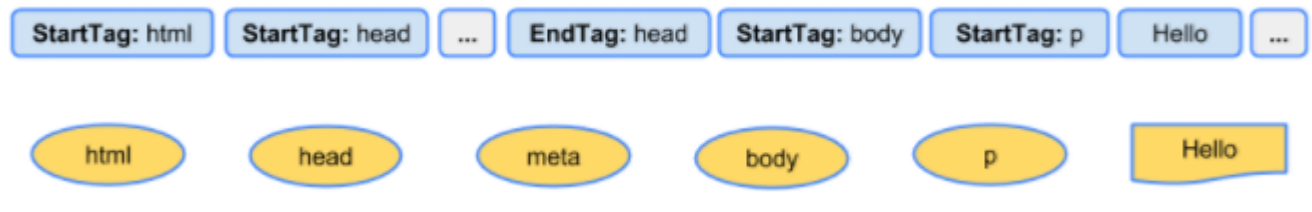


DOM



3C 62 6F 64 79 3E 48 65 6C 6C 6F 2C 20 3C 73 70 61 6E 3E 77 6F 72 6C 64 21 3C 2F 73 70 61
6E 3E 3C 2F 62 6F 64 79 3E

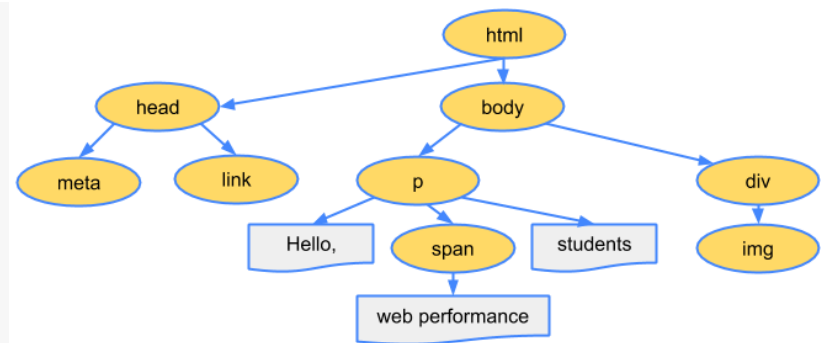
`<html><head>...</head><body><p>Hello web performance...`



DOM

O resultado final de todo esse processo é o **Document Object Model**, ou "**DOM**", da nossa página simples, que é usado pelo navegador para todos os demais processamentos da página

```
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link href="style.css" rel="stylesheet">
    <title>Critical Path</title>
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div></div>
  </body>
</html>
```



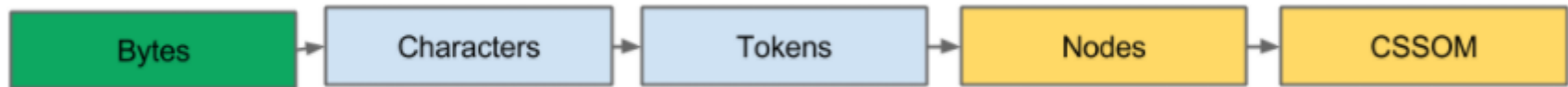
A árvore do **DOM** captura as propriedades e os relacionamentos da marcação do documento, mas não nos diz nada sobre a aparência do elemento quando renderizado. Isso é responsabilidade do **CSSOM**.

Quando o navegador estava construindo o DOM da nossa página simples, encontrou uma tag link na seção de cabeçalho do documento que referenciava uma folha de estilo CSS externa: style.css.

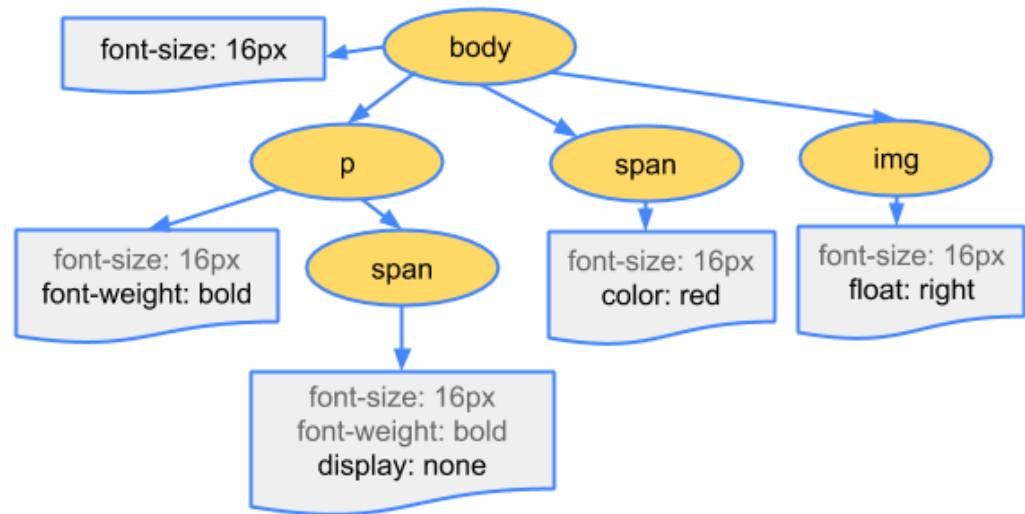
```
body { font-size: 16px }  
p { font-weight: bold }  
span { color: red }  
p span { display: none }  
img { float: right }
```

Assim como no HTML, precisamos converter as regras CSS recebidas em algo que o navegador consiga entender e usar. Portanto, repetimos o processo do HTML, mas para o CSS dessa vez.

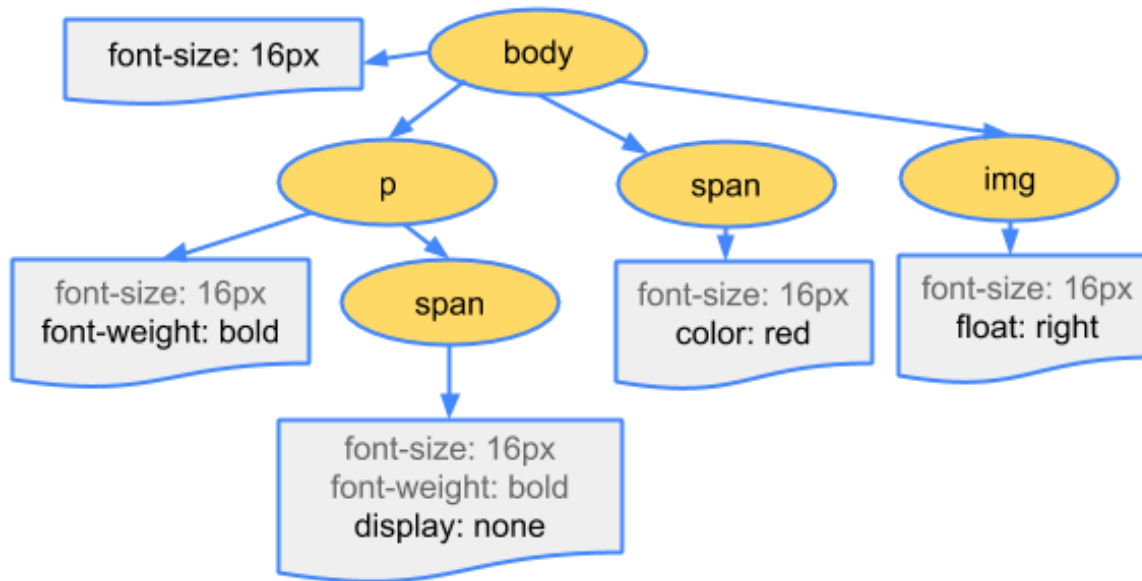
CSSOM



```
body { font-size: 16px }  
p { font-weight: bold }  
span { color: red }  
p span { display: none }  
img { float: right }
```



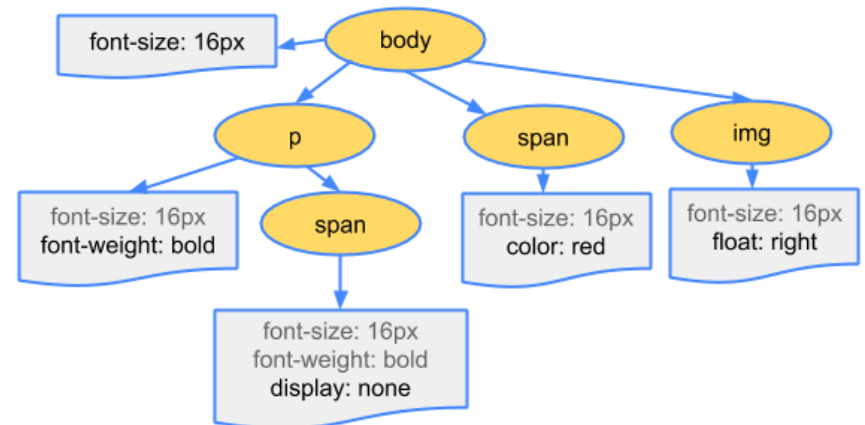
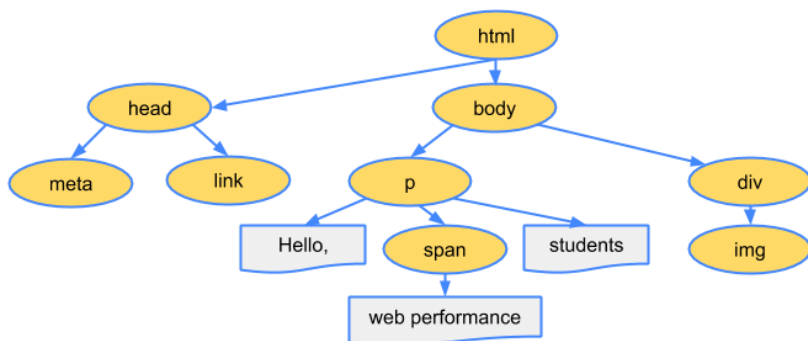
Por que o **CSSOM** tem uma estrutura de árvore? Para calcular o conjunto final de estilos de um objeto da página, o navegador começa com a regra mais geral aplicável a esse nó (por exemplo, se é secundário a um elemento "body", todos os estilos de "body" se aplicam). Em seguida, refina com recursos os estilos calculados aplicando regras mais específicas, ou seja, as regras são aplicadas em "**cascata de cima para baixo**".



Caminho crítico de renderização

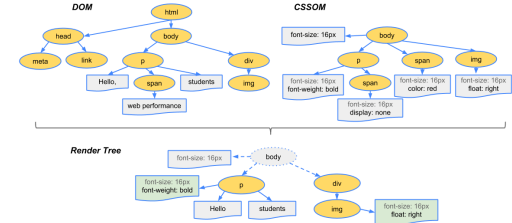
O CSSOM e o DOM são estruturas de dados independentes. O que acontece é que o navegador está escondendo uma etapa importante:

A árvore de renderização.



Render Tree


Explicação:



1. A partir da raiz da árvore DOM, percorre cada nó visível.
 - Alguns nós não são visíveis (por exemplo, tags script, tags meta e assim por diante) e são omitidos, pois não são refletidos no resultado da renderização.
 - Alguns nós foram ocultados via CSS e também são omitidos da árvore de renderização, como por exemplo, o nó "span"---do exemplo acima---não está presente na árvore de renderização porque temos uma regra explícita que define a propriedade "display: none" nela.
2. Para cada nó visível, encontre as regras do CSSOM correspondentes adequadas e aplique-as.
3. Emita nós visíveis com conteúdo e seus estilos processados.

Etapas de processamento

Recapitulando:

1. Processar a marcação HTML e criar a árvore do DOM.
 2. Processar a marcação CSS e criar a árvore do CSSOM.
 3. Combinar o DOM e o CSSOM em uma árvore de renderização.
 4. Executar o layout na árvore de renderização para calcular a geometria de cada nó.
 5. Pintar os nós individuais na tela.
- 

Bloqueio da renderização

Por padrão, o CSS é tratado como um recurso bloqueador de renderização, o que significa que o navegador não renderiza nenhum conteúdo processado até que o CSSOM seja construído. Certifique-se de manter o seu **CSS enxuto**, entregá-lo o mais rápido possível e usar tipos e consultas de mídia para desbloquear a renderização.

Na construção da árvore de renderização, vimos que o caminho crítico de renderização exige que o DOM e o CSSOM construam a árvore de renderização. Isso gera impacto importante no desempenho: tanto o HTML quanto o CSS são recursos bloqueadores de renderização. Isso é óbvio para o HTML, pois sem o DOM, não temos nada para renderizar. Mas o requisito do CSS pode ser menos evidente.

```
<link href="style.css"    rel="stylesheet">
<link href="style.css"    rel="stylesheet" media="all">
<link href="portrait.css" rel="stylesheet" media="orientation:portrait">
<link href="print.css"    rel="stylesheet" media="print">
```


E o JS na história?

JavaScript

O JavaScript é uma linguagem dinâmica executada em um navegador que permite alterar praticamente todos os aspectos do comportamento da página. Podemos modificar o conteúdo adicionando e removendo elementos da árvore do DOM; podemos modificar as propriedades do CSSOM de cada elemento; podemos lidar com as interações do usuário; entre muitas outras funções.

Ele também pode bloquear a construção do DOM e retardar a renderização da página. Para proporcionar um desempenho ótimo, faça seu JavaScript assíncrono e elimine qualquer JavaScript desnecessário do caminho crítico de renderização.

JavaScript

```
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link href="style.css" rel="stylesheet">
    <title>Critical Path: Script</title>
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div></div>
    <script>
      var span = document.getElementsByTagName('span')[0];
      span.textContent = 'interactive'; // change DOM text content
      span.style.display = 'inline'; // change CSSOM property
      // create a new element, style it, and append it to the DOM
      var loadTime = document.createElement('div');
      loadTime.textContent = 'You loaded this page on: ' + new Date();
      loadTime.style.color = 'blue';
      document.body.appendChild(loadTime);
    </script>
  </body>
</html>
```

JavaScript

Hello **interactive** students!

You loaded this page on: Wed Apr 18 2018 07:27:20 GMT-0300 (Hora oficial do Brasil)



**You're
Awesome!**

Com isso, modificamos o conteúdo e o estilo CSS de um nó DOM existente e adicionamos um nó totalmente novo ao documento.

Porém, embora o JavaScript nos dê muito poder, ele cria muitas limitações adicionais sobre como e quando a página é renderizada.


Quando o analisador HTML encontra uma tag script, interrompe seu processo de construção do DOM e passa o controle ao mecanismo do JavaScript. Depois que o JavaScript conclui a execução, o navegador reinicia do ponto de interrupção e retoma a construção do DOM.

A execução do nosso script em linha bloqueia a construção do DOM, o que, por sua vez, também retarda a renderização inicial.

Outra propriedade sutil da inclusão de scripts em nossa página é que, além de ler e modificar o DOM, eles também podem fazer o mesmo nas propriedades do CSSOM. Mas isso cria uma condição de corrida.

E se o navegador não tiver concluído o download e a criação do CSSOM no momento da execução do script? A resposta é simples, embora não muito boa para o desempenho: **o navegador interrompe a execução do script até concluir o download e a construção do CSSOM.**

Resumidamente, o JavaScript introduz uma série de novas dependências entre o DOM, o CSSOM e a execução de JavaScript. Isto pode causar atrasos significativos no navegador ao processar e renderizar a página na tela:

1. O local do script no documento é significativo.
 2. Quando o navegador encontra uma tag de script, a construção do DOM faz uma pausa até que a execução do script termine.
 3. O JavaScript pode consultar e modificar o DOM e o CSSOM.
 4. A execução do JavaScript faz uma pausa até que o CSSOM esteja pronto.
- 

JavaScript

```
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link href="style.css" rel="stylesheet">
    <title>Critical Path: Script External</title>
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div></div>
    <script src="app.js"></script>
  </body>
</html>
```

app.js

```
var span = document.getElementsByTagName('span')[0];
span.textContent = 'interactive'; // change DOM text content
span.style.display = 'inline'; // change CSSOM property
// create a new element, style it, and append it to the DOM
var loadTime = document.createElement('div');
loadTime.textContent = 'You loaded this page on: ' + new Date();
loadTime.style.color = 'blue';
document.body.appendChild(loadTime);
```


JavaScript

Adicionar a palavra-chave `async` à tag do `script` informa ao navegador para não bloquear a construção do DOM enquanto aguarda que o script seja disponibilizado, o que pode melhorar significativamente o desempenho. Mas cuidado!

```
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link href="style.css" rel="stylesheet">
    <title>Critical Path: Script Async</title>
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div></div>
    <script src="app.js" async></script>
  </body>
</html>
```

Caminho crítico de renderização

Créditos: Ilya Grigorik

<https://developers.google.com/web/fundamentals/performance/critical-rendering-path/?hl=pt-br>

Um pouco mais de Front

Front

HTML



CONTENT

CSS



PRESENTATION

JS

DYNAMIC EFFECTS/
PROGRAMMING

NOUNS

`<p></p>`
means "**paragraph**"

ADJECTIVES

`p {color: red;}`
means "the paragraph
text is **red**"

VERBS

`p.hide();`
means "**hide** the
paragraph"

Perguntas?