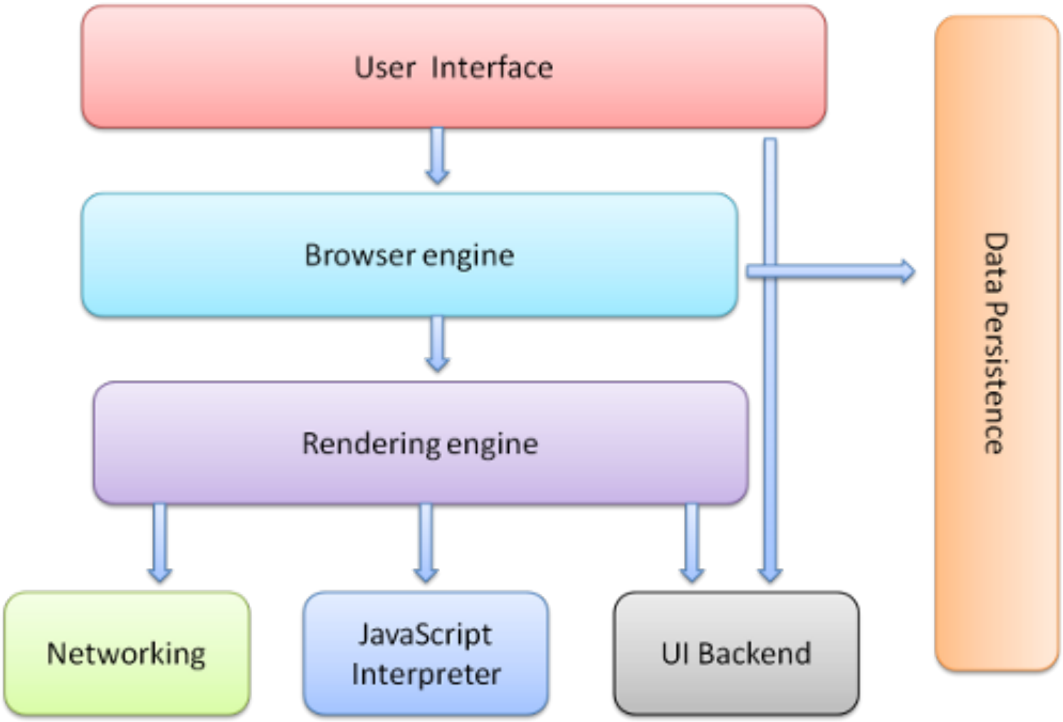
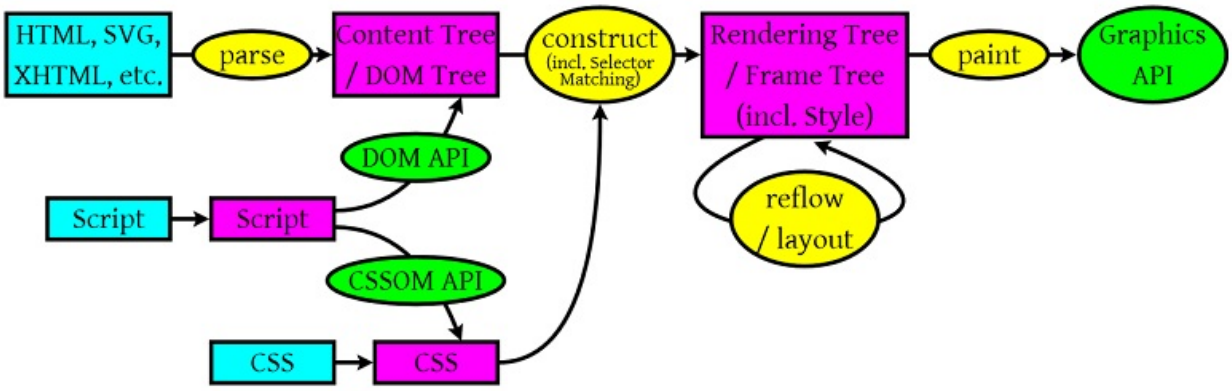
浏览器的渲染原理

浏览器架构

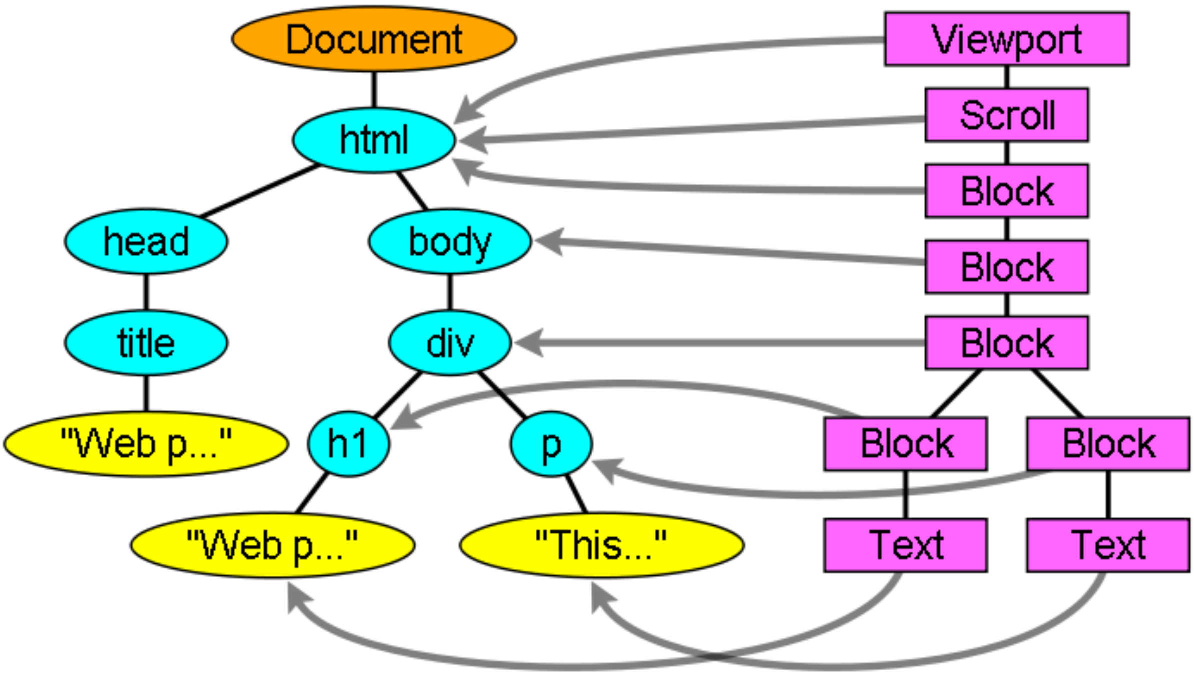


渲染流程



基本概念

1. **DOM(Document Object Model) Tree**: 由HTML文件parse生成代表内容的树状结构。
2. **CSSOM(CSS Object Model) Tree**，由CSS代码parse生成的样式规则的树状结构。
3. **Render Tree**：包含具有显示属性（颜色和大小）的长方形组成的树状结构

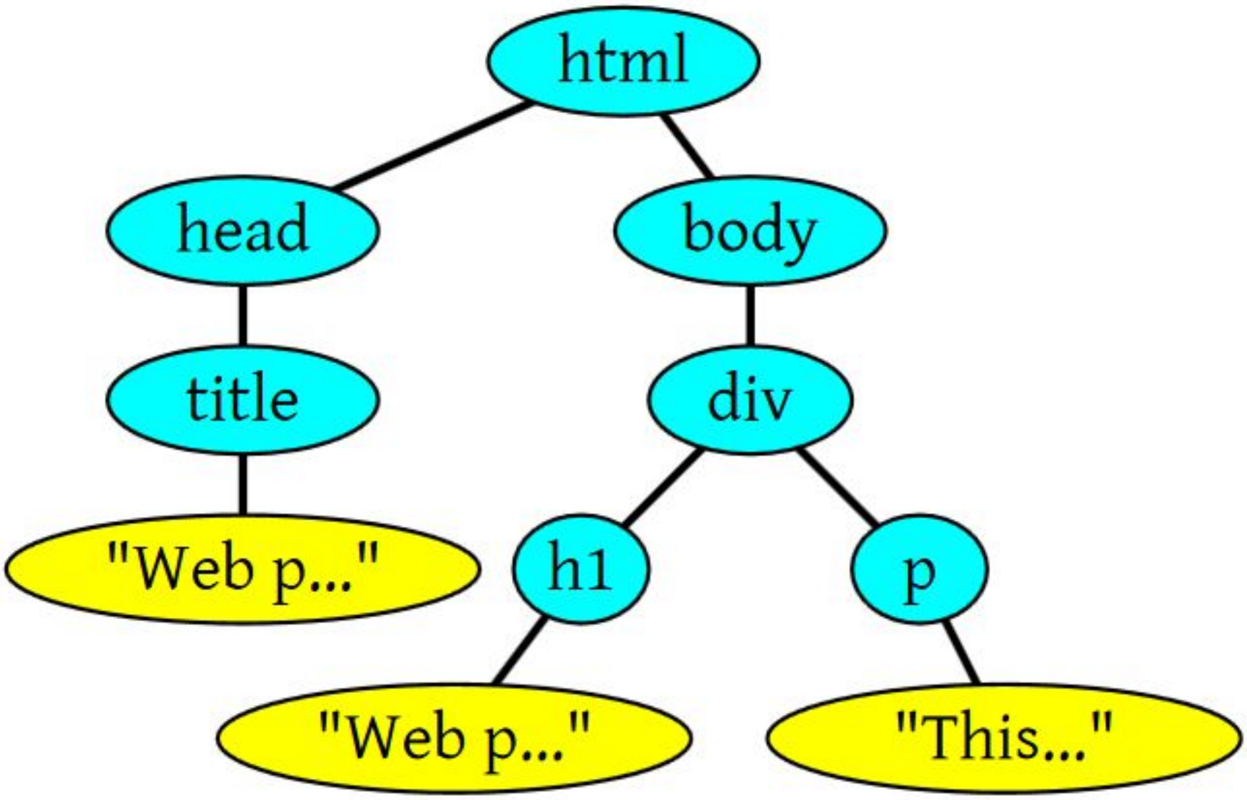


过程

1. **parse**：parse HTML/SVG/XHTML文件，产生DOM Tree；parse CSS文件生成CSS Rule Tree。
2. **construct**：DOM树和CSS规则树连接在一起construct形成Render Tree（渲染树）。
3. **reflow/layout**：计算出Render Tree每个节点的具体位置。
4. **paint**：调用系统图形API，通过显卡，将Layout后的节点内容分别呈现到屏幕上。

DOM 解析

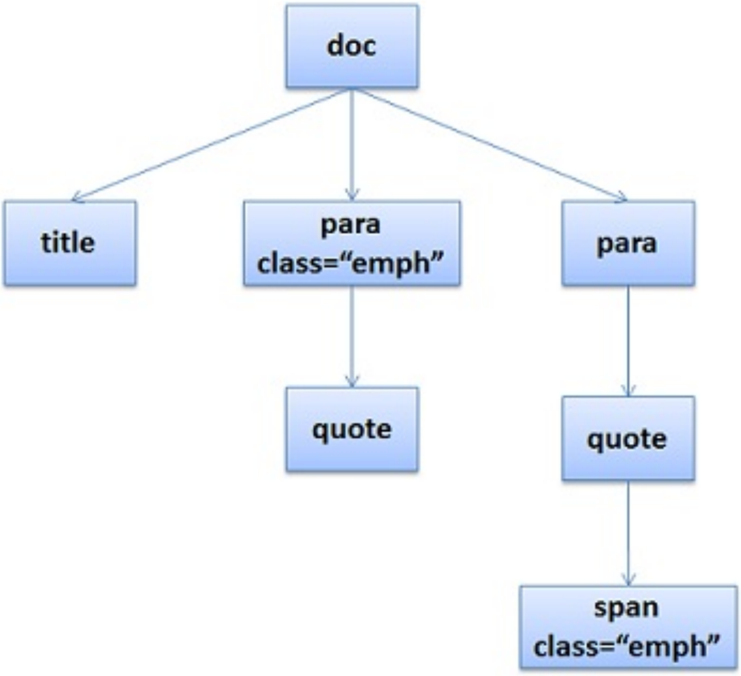
```
<html>
<html>
<head>
    <title>Web page parsing</title>
</head>
<body>
    <div>
        <h1>Web page parsing</h1>
        <p>This is an example Web page.</p>
    </div>
</body>
</html>
```



CSS 解析

样例HTML文件

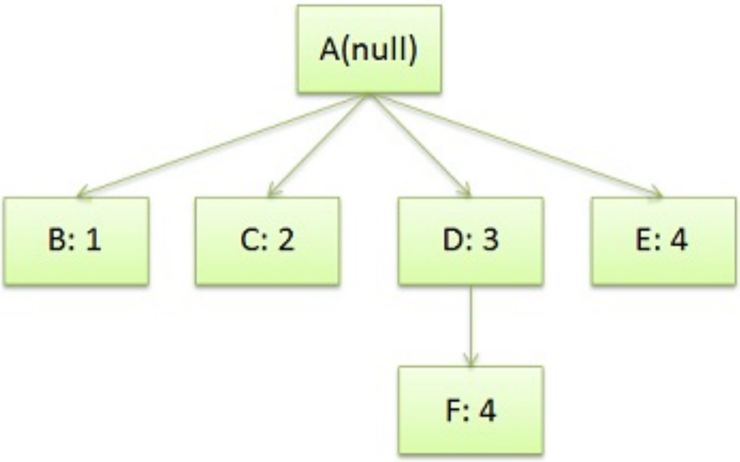
```
<doc>
<title>A few quotes</title>
<para>
  Franklin said that <quote>"A penny saved is a penny earned."</quote>
</para>
<para>
  FDR said <quote>"We have nothing to fear but <span>fear itself.</span>"
</quote>
</para>
</doc>
```



样例 DOM Tree

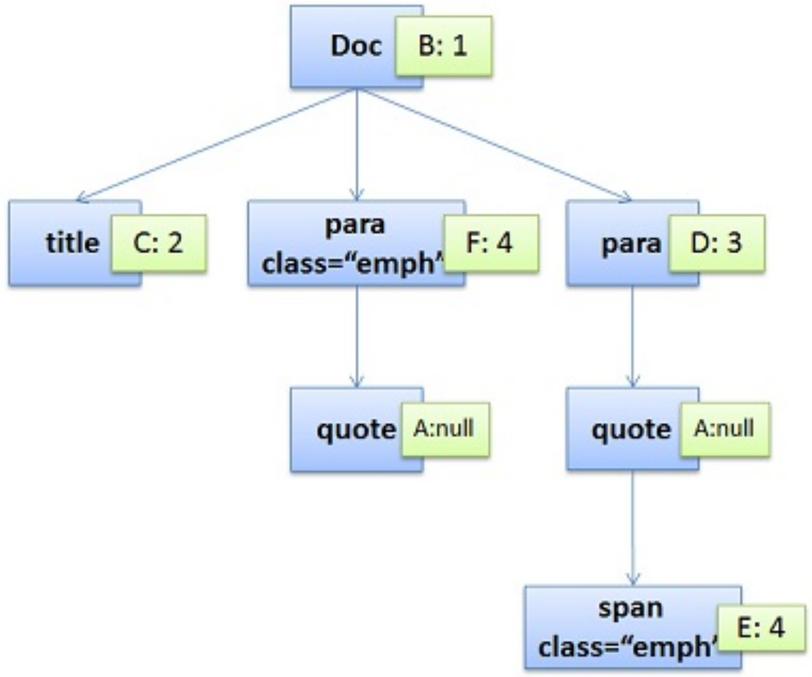
样例CSS文件

```
/* rule 1 */ doc { display: block; text-indent: 1em; }
/* rule 2 */ title { display: block; font-size: 3em; }
/* rule 3 */ para { display: block; }
/* rule 4 */ [class="emph"] { font-style: italic; }
```



样例CSS Tree

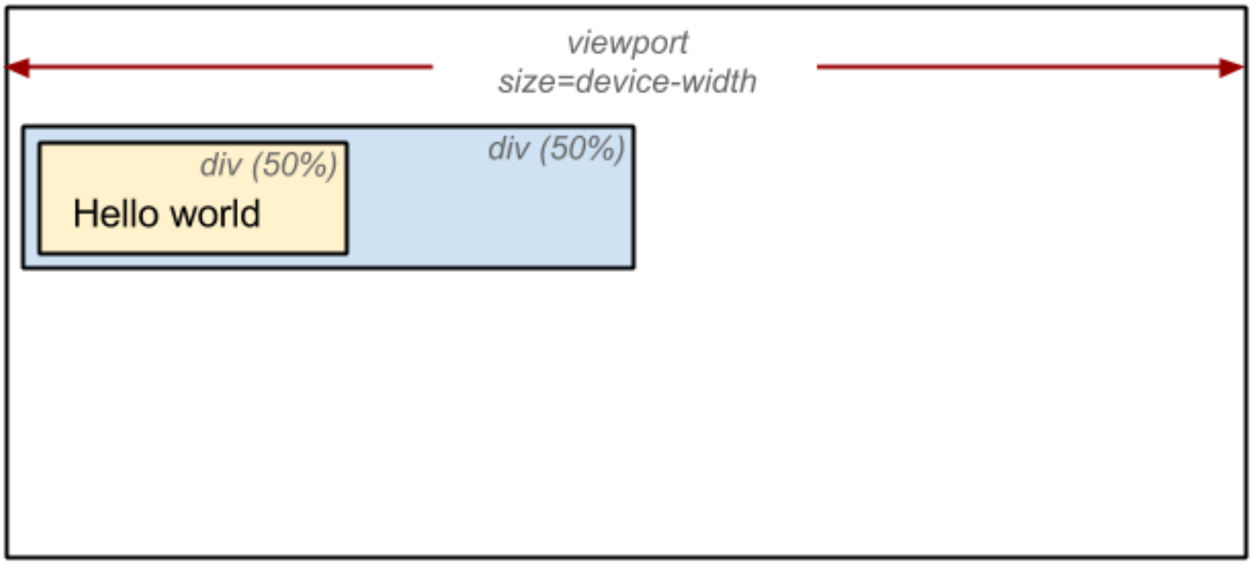
样例Content Tree（同时包含内容和样式信息）



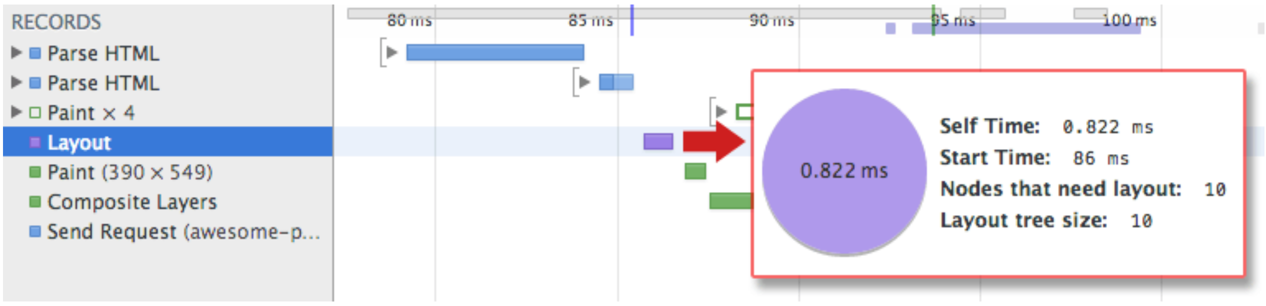
Reflow/Layout 过程

布局的过程

1. parent渲染对象决定它的宽度
2. parent渲染对象读取chilidren，并： a. 放置child渲染对象（设置它的x和y） b. 在需要时（它们当前为dirty或是处于全局layout或者其他原因）调用child渲染对象的layout，这将计算child的高度
3. parent渲染对象使用child渲染对象的累积高度，以及margin和padding的高度来设置自己的高度－这将被parent渲染对象的parent使用
4. 将dirty标识设置为false



The output of the layout process is a "box model," which precisely captures the exact position and size of each element within the viewport: all of the relative measurements are converted to absolute pixels on the screen.



Self Time: 0.822 ms
Start Time: 86 ms
Nodes that need layout: 10
Layout tree size: 10

- The "Layout" event captures the render tree construction, position, and size calculation in the Timeline.
- When layout is complete, the browser issues "Paint Setup" and "Paint" events, which convert the render tree to pixels on the screen.

Dirty bit系统

为了不因为每个小变化都全部重新布局，浏览器使用一个dirty bit系统，一个渲染对象发生了变化或是被添加了，就标记它及它的children为dirty——需要layout。存在两个标识——dirty及children are dirty，children are dirty说明即使这个渲染对象可能没问题，但它至少有一个child需要layout。

Repaint 和 Reflow

- Repaint——屏幕的一部分要重画，比如某个CSS的背景色变了。但是元素的几何尺寸没有变。
- Reflow——意味着元件的几何尺寸变了，我们需要重新验证并计算Render Tree。是Render Tree的一部分或全部发生了变化。这就是Reflow，或是Layout。（**HTML使用的是flow based layout，也就是流式布局，所以，如果某元件的几何尺寸发生了变化，需要重新布局，也就叫reflow**）reflow 会从这个root frame开始递归往下，依次计算所有的结点几何尺寸和位置，在reflow过程中，可能会增加一些frame，比如一个文本字符串必需被包装起来。

用JS完成交互

script in html

```
<!DOCTYPE html>
<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><img src="awesome-photo.jpg"></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>
```

This can cause the browser significant delays in processing and rendering the page on the screen:

- The location of the script in the document is significant.
- When the browser encounters a script tag, DOM construction pauses until the script finishes executing.
- JavaScript can query and modify the DOM and the CSSOM.
- JavaScript execution pauses until the CSSOM is ready.

script in js file

```
<!DOCTYPE html>
<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><img src="awesome-photo.jpg"></div>
    <script src="app.js"></script>
  </body>
</html>
```

```
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);
```

async

Adding the async keyword to the script tag tells the browser not to block DOM construction while it waits for the script to become available, which can significantly improve performance.

```html
<!DOCTYPE html>
<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><img src="awesome-photo.jpg"></div>
    <script src="app.js" async></script>
  </body>
</html>
```

原文链接

https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction?hl=en

https://kb.cnblogs.com/page/129756/