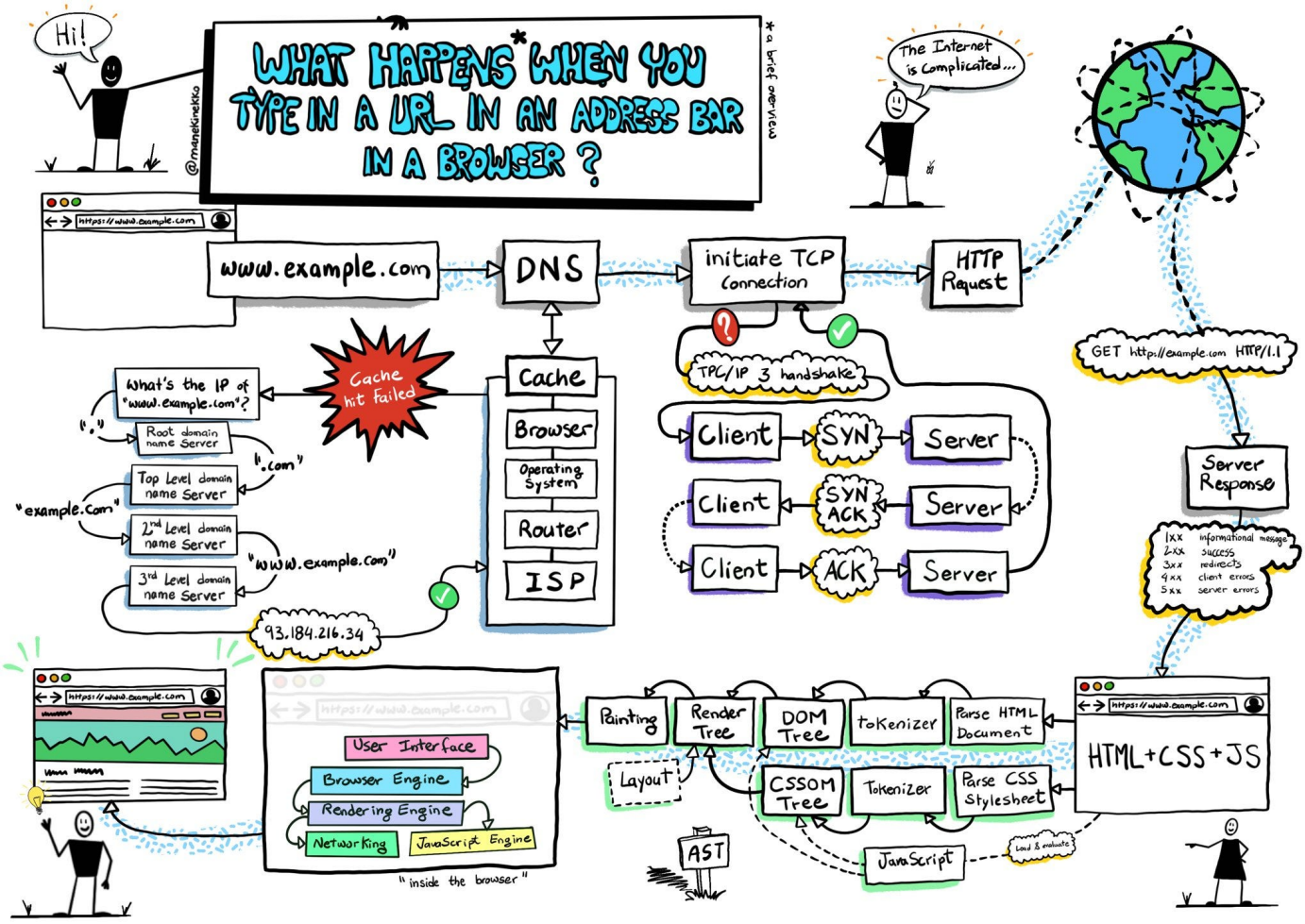
02_coderwhy前端八股文（二） - 浏览器渲染

01-浏览器输入一个URL并enter时，到底发生了什么?

When you type a URL and press enter what is the process behind the page loading

当你在浏览器中输入一个URL比如www.example.com时，我们需要先找到它对应的IP地址：这个过程被称为DNS解析，即域名系统解析。



DNS（Domain Name System）服务器解析

缓存查找过程：

- **浏览器缓存**：首先，浏览器会检查它的缓存中是否有这个域名的记录，因为之前访问过的网址的解析结果可能会被存储在浏览器缓存中。
- **操作系统缓存**：如果浏览器缓存中没有找到，浏览器会询问操作系统，因为操作系统也可能有自己的DNS缓存。
- **路由器缓存**：如果操作系统中也没有找到，请求会发送到本地网络的路由器，它同样可能有自己的DNS缓存。
- **ISP（ Internet service provider）缓存**：如果以上都没有缓存记录，请求最终会发送到你的互联网服务提供商（ISP），它们通常会有更大范围的DNS缓存。

DNS递归解析

如果所有本地缓存查找都失败，DNS查询就变成了一个递归查询过程，涉及到多个DNS服务器：

- **根域名服务器**：首先，你的DNS查询会被发送到根域名服务器。根服务器是最高级别的DNS服务器，负责重定向到负责管理顶级域名（如.com、.net等）的顶级域名服务器。
- **顶级域名服务器（TLD服务器）**：根服务器会告诉你的ISP的DNS服务器去查询哪个顶级域名服务器来找到`.com`域的信息。这个服务器掌握所有`.com`域名及其相应服务器的信息。
- **权威域名服务器**：一旦你的DNS查询到达了正确的顶级域名服务器，它会进一步定向到负责`example.com`的权威服务器。权威服务器有该域名对应的具体IP地址。

IP地址的获取

最终，权威域名服务器会提供`www.example.com`域名对应的IP地址（如图中的93.184.216.34），这个信息会被发送回用户的电脑。

缓存结果

一旦IP地址被找到，它通常会被存储在浏览器、操作系统、路由器或ISP的DNS缓存中，以便未来的查询可以更快得到解析。

TCP/HTTP请求

TCP (Transmission Control Protocol, 传输控制协议)，TCP是一种面向连接的协议，用于在网络中的两个端点之间建立可靠的会话。

以下是TCP连接建立过程，通常称为三次握手（TCP 3-way handshake）：

SYN（Synchronize）：

- 客户端发送一个SYN包到服务器以初始化一个连接。
- 客户端设置一个随机的序列号，告诉服务器它准备开始发送数据。
- 序列号不仅仅是在握手期间使用的，后续传输数据也会用到，用来保证数据的完整性和顺序。

SYN-ACK（Synchronize-Acknowledgment）：

- 服务器接收到客户端的SYN包后，会发送一个SYN-ACK包作为响应。
- 服务器同样设置一个随机的序列号，并将客户端的序列号加一，发送回给客户端，确认已经收到了客户端的同步请求（+1表示服务器确认收到）。

ACK（Acknowledgment）：

- 客户端收到服务器的SYN-ACK后，发送一个ACK包作为回应。
- 这个ACK包将服务器的序列号加一，并可能包含客户端准备发送的数据的开始部分（比如HTTP请求行`GET / HTTP/1.1`和请求头，这个被称之为TCP快速打开）。
- 此时，TCP连接已经建立，双方可以开始数据传输。

HTTP (Hypertext Transfer Protocol, 超文本传输协议)，它是建立在TCP连接之上的应用层协议。（这里不展开网络的分层架构）

HTTP工作流程如下：

客户端请求：

- 一旦TCP连接建立，客户端（通常是Web浏览器）就可以通过这个连接发送一个HTTP请求到服务器。
- 这个请求包含了方法（GET、POST等）、URI（统一资源标识符）和协议版本，以及可能包含的请求头和请求体。

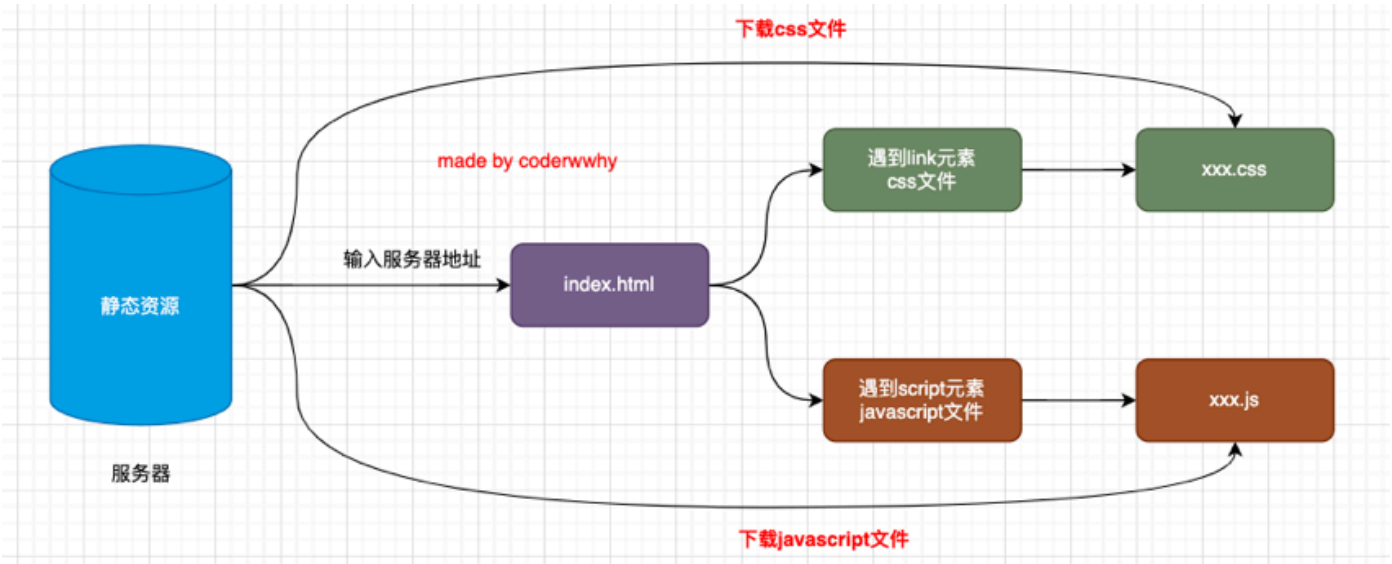
服务器响应：

- 服务器接收到HTTP请求后，会处理这个请求并返回一个HTTP响应。
- 响应通常包括一个状态码（如200表示成功，404表示未找到），响应头，以及任何响应内容（如请求的HTML文件）。

TCP为HTTP提供了一个可靠的通道，确保数据正确、完整地从服务器传输到客户端。

浏览器渲染过程

服务器下载资源的过程：



那么当一个页面被下载下来后它是如何被渲染的呢?

这里我们需要先学习一个重要的知识：浏览器内核。

我们经常说的浏览器内核指的是浏览器的排版引擎：

- 排版引擎（layout engine），也称为浏览器引擎（browser engine）、页面渲染引擎（rendering engine）或样版引擎。
- 也就是一个网页下载下来后，就是由我们的渲染引擎来帮助我们解析的。

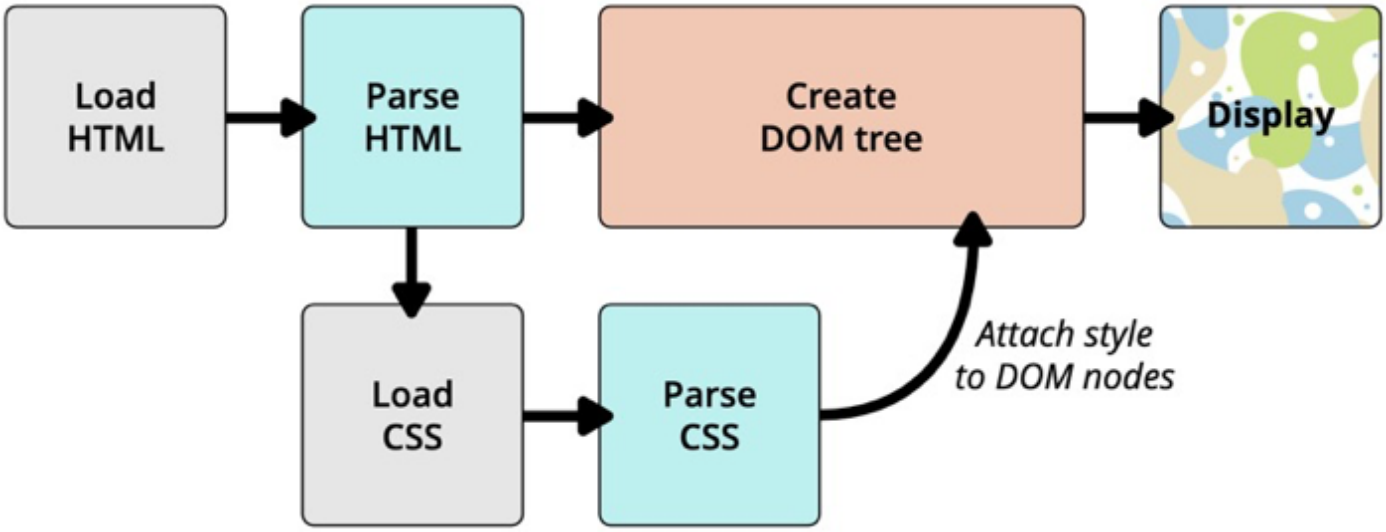
常见的浏览器内核有哪些呢?

常见的浏览器内核有

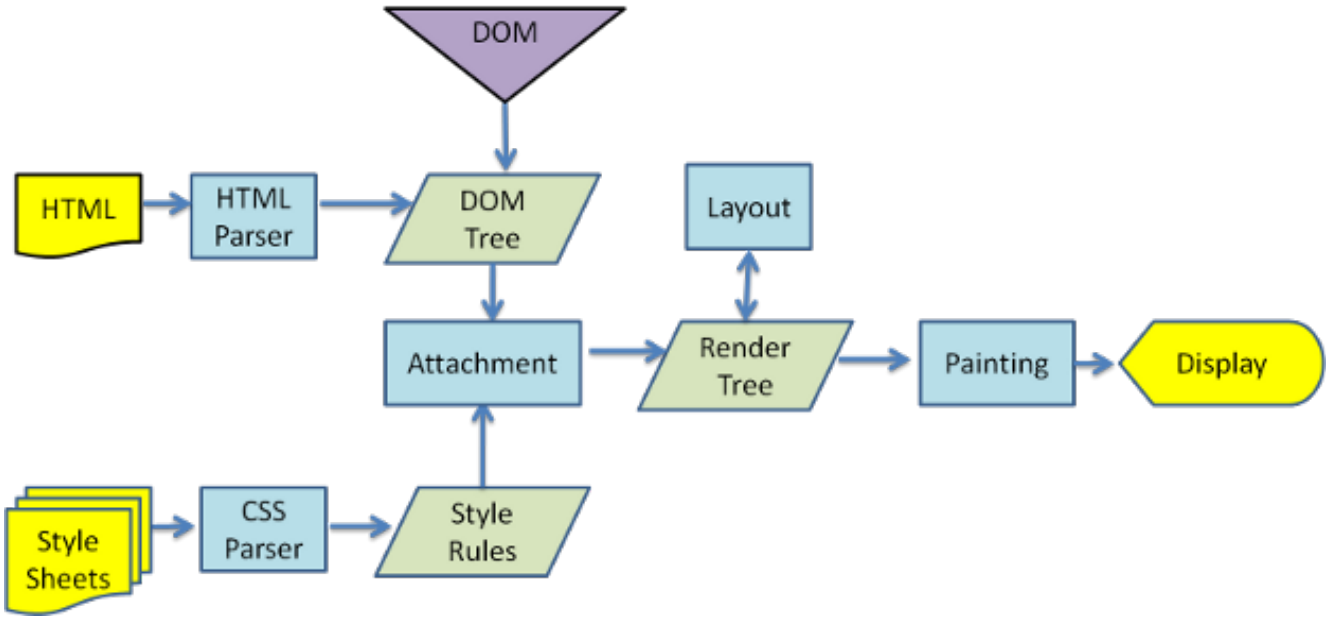
- Trident （三叉戟）：IE、早期的360安全浏览器、早期的搜狗高速浏览器、早期的百度浏览器、早期的UC浏览器；（微软已经放弃）
- Gecko（壁虎）：Mozilla Firefox；
- Presto（急板乐曲）-> Blink （眨眼）：Opera
- Webkit：Safari、移动端浏览器（Android、iOS）

- Webkit -> Blink： Google Chrome，Edge，360极速浏览器，搜狗高速浏览器

渲染引擎在拿到一个页面后，如何解析整个页面并且最终呈现出我们的网页呢?



更加详细的过程：

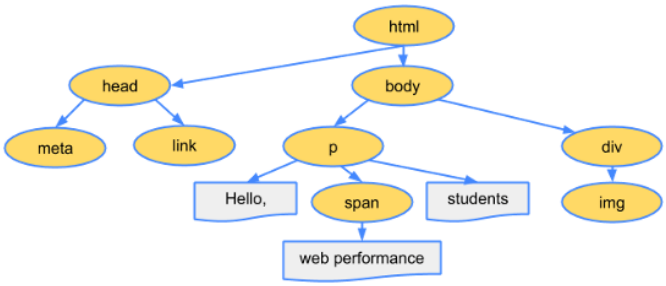


解析一：HTML解析过程

因为默认情况下服务器会给浏览器返回index.html文件，所以解析HTML是所有步骤的开始。

解析HTML，会构建DOM Tree：

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <p>Hello <span>web performance</span> students!</p>
  <div>
    <img src="awesome-photo.jpg">
  </div>
</body>
</html>
```

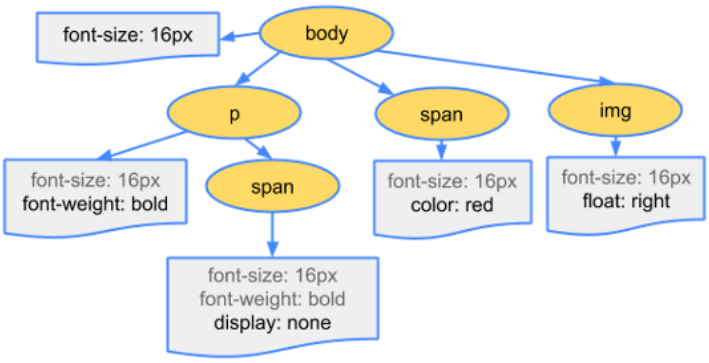


解析二 – 生成CSS规则

在解析的过程中，如果遇到CSS的link元素，那么会由浏览器负责下载对应的CSS文件：

- 注意：下载CSS文件是不会影响DOM的解析的；
- 浏览器下载完CSS文件后，就会对CSS文件进行解析，解析出对应的规则树：
- 我们可以称之为 CSSOM（CSS Object Model，CSS对象模型）；

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

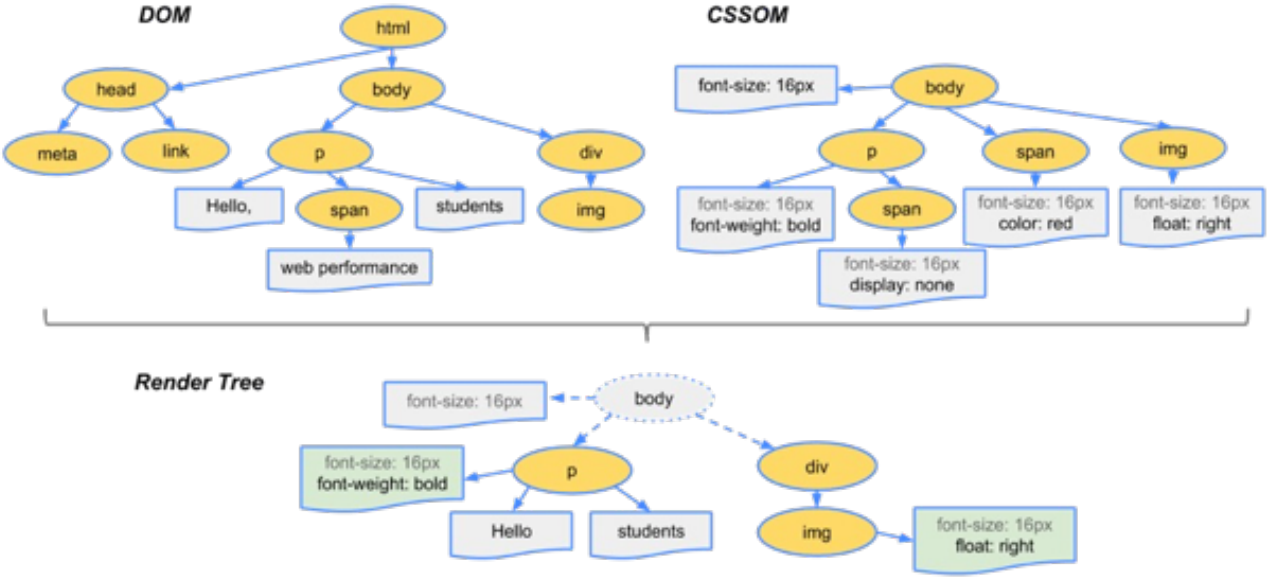


解析三 – 构建Render Tree

注意一：link元素不会阻塞DOM Tree的构建过程，但是会阻塞Render Tree的构建过程

- 这是因为Render Tree在构建时，需要对应的CSSOM Tree；

注意二：Render Tree和DOM Tree并不是一一对应的关系，比如对于display为none的元素，压根不会出现在render tree中；



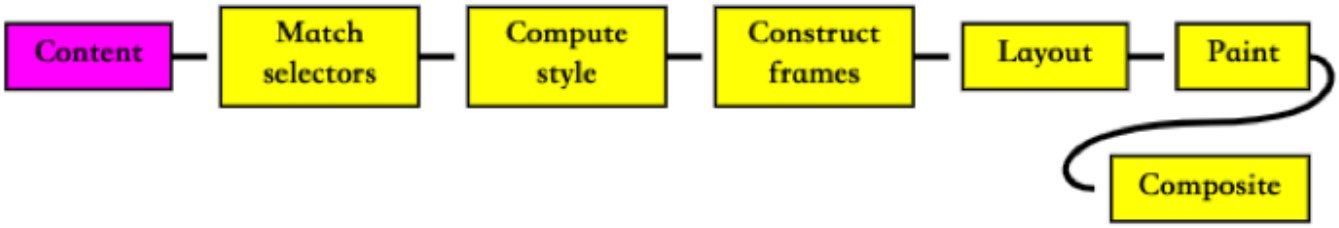
解析四 – 布局（layout）和绘制（Paint）

第四步是在渲染树（Render Tree）上运行布局（Layout）以计算每个节点的几何体。

- 渲染树会表示显示哪些节点以及其他样式，但是不表示每个节点的尺寸、位置等信息；
- 布局是确定呈现树中所有节点的宽度、高度和位置信息；

第五步是将每个节点绘制（Paint）到屏幕上

- 在绘制阶段，浏览器将布局阶段计算的每个frame转为屏幕上实际的像素点；
- 包括将元素的可见部分进行绘制，比如文本、颜色、边框、阴影、替换元素（比如img）



Match selectors：浏览器遍历CSSOM，将选择器与DOM树中的元素匹配。这个过程决定了哪些CSS规则应用于哪些DOM元素。

Compute style：在选择器匹配后，浏览器计算每个元素的最终样式。这包括计算具体的样式值，处理继承的样式以及解析因层叠产生的任何冲突。

Construct frames：这通常是指生成布局树，它是渲染树的一部分，仅包含要布局和绘制的元素。这一步骤涉及确定文档的结构层次和包含块。

- 布局树和渲染树是有微小的差异，布局树是渲染树的子集，不包含渲染树中元素的颜色、背景、阴影等信息

回流和重绘

理解回流reflow：（也可以称之为重排）

- 第一次确定节点的大小和位置，称之为布局（layout）。
- 之后对节点的大小、位置修改重新计算称之为回流。

什么情况下引起回流呢?

- 比如DOM结构发生改变（添加新的节点或者移除节点）；
- 比如改变了布局（修改了width、height、padding、font-size等值）
- 比如窗口resize（修改了窗口的尺寸等）
- 比如调用getComputedStyle方法获取尺寸、位置信息；

理解重绘repaint：

- 第一次渲染内容称之为绘制（paint）。
- 之后重新渲染称之为重绘。

什么情况下会引起重绘呢?

- 比如修改背景色、文字颜色、边框颜色、样式等；
- 比如修改阴影，box-shadow、text-shadow等；
- 比如修改背景图像：background-image等

回流一定会引起重绘，所以回流是一件很消耗性能的事情。

所以在开发中要尽量避免发生回流：

- 1.修改样式时尽量一次性修改
  - 比如通过cssText修改，比如通过添加class修改
- 2.尽量避免频繁的操作DOM
  - 我们可以在一个DocumentFragment或者父元素中将要操作的DOM操作完成，再一次性的操作；
- 3.尽量避免通过getComputedStyle获取尺寸、位置等信息；
  - 频繁调用`getComputedStyle`可以导致回流，因为浏览器需要提供准确的计算值。
- 4.对某些元素使用position的absolute或者fixed
  - 并不是不会引起回流，而是开销相对较小，不会对其他元素造成影响。

composite合成

绘制的过程，可以将布局后的元素绘制到多个合成图层中。

- 这是浏览器的一种优化手段；

默认情况下，标准流中的内容都是被绘制在同一个图层（Layer）中的；

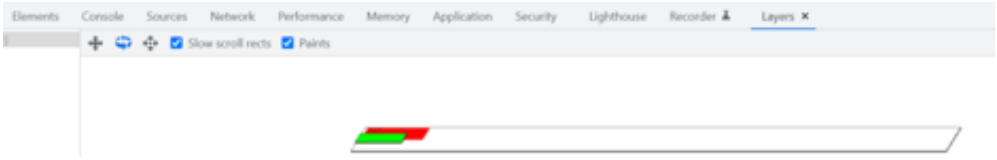
而一些特殊的属性，会创建一个新的合成层（ CompositingLayer ），并且新的图层可以利用GPU来加速绘制；

- 因为每个合成层都是单独渲染的；

那么哪些属性可以形成新的合成层呢？常见的一些属性：

- 3D transforms
- video、canvas、iframe
- opacity 动画转换时；
- position: fixed
- will-change：一个实验性的属性，提前告诉浏览器元素可能发生哪些变化；
- animation 或 transition 设置了opacity、transform；

分层确实可以提高性能，但是它以内存管理为代价，因此不应作为 web 性能优化策略的一部分过度使用。



面试回答

在用户在浏览器中输入一个URL，并且按下enter键时，里面包含了非常多的技术细节。

第一，DNS解析：

- 用户输入的URL通常会是一个域名地址，直接通过域名是无法找到服务器的，因为服务器本质上是一套拥有IP地址的主机。
- 我们需要通过DNS服务器来解析域名，并且获取IP地址。
- DNS会查找缓存，缓存的查找包括浏览器缓存、操作系统缓存、路由器缓存、ISP缓存，如果在缓存中找到就可以使用对应的IP地址去连接主机。
- 如果缓存查找失败，我们需要通过DNS递归解析，解析过程包括根域名服务器、顶级域名服务器、权威域名服务器。
- 最终找到IP地址，就可以通过该IP地址去连接服务器，并且IP地址信息会被发送回用户的电脑，缓存起来。

第二，TCP连接：

- 虽然我们是发送的HTTP请求，但是HTTP协议是应用层协议，它是建立在TCP传输层协议之上的，所以我们需要先进行TCP连接。
- TCP的连接会经常三次握手，客户端发送SYN包，服务器接收到后返回一个SYN-ACK包，客户端再次发送一个ACK包，完成握手过程。
- 此时TCP连接建立完成，双方就可以开始传输数据了。

第三，HTTP请求：

- 一旦TCP建立连接成功，客户端就可以通过这个链接发送HTTP请求，包括请求方法、URI、协议版本、请求头、请求体。
- 服务器收到HTTP请求后，会处理这个请求，并且返回一个HTTP响应。
- HTTP响应包括状态码、响应头、响应内容，我们这里请求的通常是index.html文件。

第四，HTML解析和CSS解析：

- 浏览器在获取到index.html后可以开始对文档进行解析。
- 包括HTML解析来构建DOM Tree，在这个过程中它会遇到CSS文件和JS文件。
- 遇到CSS和JS文件会继续向服务器发送HTTP请求，并且下载CSS、JS文件。
- 之后对CSS文件进行解析，解析出对应的CSSOM（CSS Object Model）。

- JS文件会由JavaScript引擎来执行，我们后续可以再讨论。

第五，渲染render、布局layout、绘制paint：

- DOM Tree和CSSOM可以共同来构建Render Tree。
- 之后会在Render Tree上运行布局layout计算每个阶段的几何体。
- 再由浏览器将每个阶段绘制paint到屏幕上的像素点。

第六，composite合成：

- 这里还有一个优化的手段是将元素绘制到多个合成图层中。
- 默认情况下，标准流中的内容是被绘制到同一个图层（Layer）中的。
- 我们可以通过一些特定的方法来创建新的合成层（CompositingLayer），新的图层可以利用GPU来加速绘制。
- 比如我们通过3D Transforms/video/canvas/iframe/positionfixed/will-change等
- 但是分层确实可以提供性能，但是它是以内存管理为代价的。

这样用户最终就看到了我们屏幕上显示的网页。当然这个过程中还有更多的细节，包括重绘、回流的问题，包括JavaScript的执行过程、JavaScript引擎（比如V8引擎）等知识，我这边也可以继续进行扩展讨论。

02-描述浏览器的基本功能以及它是如何运用和加载网页的

浏览器是操作系统与网页应用程序之间的桥梁，其实也是软件工程中的分层架构（引出了软件工程的分层架构）。

- 从用户的角度来说，浏览器可以帮助他们请求网页，并且接下解析，运行，让他们可以快速浏览器自己需要的应用内容。
- 从开发者的角度来说，浏览器为我们开发的页面提供了运行环境，并且我们按照浏览器的规则去编写的网站最终可以很好的给用户呈现。

它具体的请求和解析网页的过程是：参考01-浏览器输入一个URL并enter时，到底发生了什么?

03-浏览器内核是什么？常见的浏览器内核有哪些?

我们经常说的浏览器内核指的是浏览器的排版引擎：

- 排版引擎（layout engine），也称为浏览器引擎（browser engine）、页面渲染引擎（rendering engine）或样版引擎。
- 也就是一个网页下载下来后，就是由我们的渲染引擎来帮助我们解析的。

常见的浏览器内核主要包括（从优先级的依次说）：

1. **Blink**：由Google开发，是Chrome浏览器以及最新版本的微软 Edge浏览器的内核。Blink是WebKit的一个分支。
2. **WebKit**：最初由苹果公司为Safari浏览器开发。它是开源的渲染引擎，也被其他一些浏览器采用，如iOS上的所有浏览器。
3. **Gecko**：由Mozilla开发，是Firefox浏览器的内核。
4. **Trident**：由微软开发，曾经是Internet Explorer浏览器的内核。随着新版Edge浏览器转向Blink内核，Trident的使用已经大幅减少，微软也已经放弃对它的维护更新。
5. 国内大多数尝试在过去使用的往往是Trident内核，目前大多数尝试都转向了使用Blink内核。

04-描述浏览器解析并且展示HTML、CSS和JavaScript的基本过程

- 参考：01-浏览器输入一个URL并enter时，到底发生了什么?

JavaScript执行过程后续会讲解。

05-定义回流（Reflow）在浏览器渲染过程中的意义，并解释何时回触发回流?

回流是浏览器渲染过程中的一个阶段，涉及计算所有元素的位置和大小。

- 当DOM的结构发生任何改变时（例如元素的添加、移除、移动或大小变化），浏览器需要重新计算元素的几何属性，然后确定它们在页面上的确切位置。
- 这个过程是必需的，因为页面布局是动态的，元素的变化可能影响其它元素的布局。
- 但是我们要尽量避免或者减少回流的发生，因为这个过程也是非常消耗性能的。

什么情况下引起回流呢?

- 比如DOM结构发生改变（添加新的节点或者移除节点）；
- 比如改变了布局（修改了width、height、padding、font-size等值）
- 比如窗口resize（修改了窗口的尺寸等）
- 比如调用getComputedStyle方法获取尺寸、位置信息；
- 等等

回流是一个计算密集的过程，可能会影响到网页的性能，特别是在复杂的页面布局中。

- 因此，优化网页以减少不必要的回流是提高性能的重要策略和手段。
- 这包括尽量减少在一个操作过程中对DOM的多次修改，利用CSS的类进行样式变更等。
- 这也是现代框架Vue、React的源码内部经常会涉及到的优化手段。（这里可能引出进一步问你Vue、React源码的问题）。

06-解释什么是重绘（Repaint）以及它在浏览器渲染网页时的作用。

重绘是浏览器渲染过程中的一个步骤，它涉及到更新页面中元素的视觉表现，但不涉及这些元素的布局位置。

- 重绘发生在元素的外观变化时，如改变颜色、阴影或者透明度等。
- 这些变化不会影响到元素的几何结构（即位置和形状）。

什么情况下会引起重绘呢?

- 比如修改背景色、文字颜色、边框颜色、样式等；
- 比如修改阴影，box-shadow、text-shadow等；
- 比如修改背景图像：background-image等

回流一定会引起重绘，所以回流是一件很消耗性能的事情。虽然重绘不涉及布局的变化，因此比回流成本低，但是也会对性能造成负面影响。

我们开发中可以通过下面的方案来进行优化：

- **合并视觉变更**：尽可能在一次操作中合并多个视觉变更，以减少重绘的次数。
- **新的合成层（CompositingLayer）**：在特定情况下，可以创建新的合成层，并且新的图层可以利用GPU来加速绘制也可以提供性能。

07-分析将JavaScript文件防止在HTML文档的不同位置（如头部和尾部）对加载和执行的影响。

这个主要考察JavaScript的下载和执行会阻塞DOM Tree的构建，另外你在回答的时候可以主动引出defer、async，显示你知识面的深度和广度。

通常在开发中，JavaScript元素的编写位置我们会放在头部或者尾部，它们是会有区别的。

情况一：普通JavaScript放在头部

- 当JavaScript文件放置在文档的头部时，浏览器在解析HTML到达`<script>`标签时会暂停，进行脚本的下载和执行。
- 这意味着，直到JavaScript执行完成，页面的其余部分（如HTML和CSS）才会继续加载。
- 这种做法通常会导致可见的延迟，尤其是当脚本文件较大或网络条件较差时。
- 另外如果涉及到DOM元素操作，因为DOM还没有构建完成，所以操作DOM可能会失败，也需要慎重操作。

情况二：普通JavaScript放到尾部

- 将JavaScript放在页面底部是一种常见的做法，以提高页面的加载速度。
- 这样，浏览器可以先加载页面的所有HTML和CSS内容，使用户尽快看到页面的结构和样式，而脚本将在页面的内容完全加载之后才开始下载和执行。
- 这种方法通常可以提高首次渲染时间和用户感知的加载速度。
- 当JavaScript放置在页面的底部时，它将在大部分或全部DOM元素已经加载后执行，从而减少了因DOM元素尚未加载而导致的错误。

我们可以结合前面学习defer和async来回答：

- 为了进一步提升性能，现代的开发实践中常常利用`async`和`defer`属性。
- `defer`属性让脚本的下载与DOM解析同步进行，但延迟到整个页面解析完成后再执行。
- `async`属性允许浏览器异步下载脚本，而不阻塞DOM的解析，脚本会在下载完后立即执行。
- 这两种属性都有助于优化加载时间和用户体验，选择使用哪一种取决于脚本的具体作用和需求。

08-解释CSS文件是如何被浏览器解析并应用到网页上的。

下载CSS文件：当浏览器遇到一个 `<link>` 标签或 `@import` 指令引用的CSS文件时，它会首先下载这个文件。

解析CSS：下载完成后，浏览器会解析CSS文件内容，将CSS代码转换成浏览器可以理解和使用的结构。

- 这一结构通常称为CSS对象模型（CSSOM）；

构建渲染树：当DOM和CSSOM都构建完毕后，浏览器将这两者结合起来形成渲染树。

- 渲染树只包含实际要渲染的节点，并且每个节点都有相应的CSS样式信息。

布局layout：一旦渲染树构建完成，浏览器会进行布局。

- 在这个过程中，浏览器会计算每个节点的确切位置和大小。布局的结果依赖于渲染树中的节点以及它们的样式。

绘制Paint：布局完成后，下一步是绘制，即浏览器将渲染树中的每个节点转换为屏幕上的实际像素。

- 这包括绘制文本、颜色、图像、边框等视觉效果。

合成Composite：某些元素可能需要在单独的层上进行绘制和合成。

- 单独的合成层通常用于优化性能，特别是在动画和高频变化的元素上，因为它们可以在不影响页面其他部分的情况下独立更新。

09-详述浏览器加载和执行JavaScript文件的机制。

JavaScript的执行相关的等到后续讲JavaScript面试题、JavaScript引擎、V8引擎等内容时详细讲解。