

## 06 | 渲染流程（下）：HTML、CSS和JavaScript，是如何变成页面的？

2019-08-17 李兵

《浏览器工作原理与实践》

课程介绍 >



讲述：李兵

时长 12:45 大小 11.68M



在 [上篇文章](#) 中，我们介绍了渲染流水线中的 **DOM 生成**、**样式计算**和**布局**三个阶段，那今天我们接着讲解渲染流水线后面的阶段。

这里还是先简单回顾下上节前三个阶段的主要内容：在 HTML 页面内容被提交给渲染引擎之后，渲染引擎首先将 HTML 解析为浏览器可以理解的 DOM；然后根据 CSS 样式表，计算出 DOM 树所有节点的样式；接着又计算每个元素的几何坐标位置，并将这些信息保存在布局树中。

### 分层

现在我们有布局树，而且每个元素的具体位置信息都计算出来了，那么接下来是不是就要开始着手绘制页面了？



答案依然是否定的。

因为页面中有很多复杂的效果，如一些复杂的 3D 变换、页面滚动，或者使用 z-indexing 做 z 轴排序等，为了更加方便地实现这些效果，渲染引擎还需要为特定的节点生成专用的图层，并生成一棵对应的图层树（LayerTree）。如果你熟悉 PS，相信你会很容易理解图层的概念，正是这些图层叠加在一起构成了最终的页面图像。

要想直观地理解什么是图层，你可以打开 Chrome 的“开发者工具”，选择“Layers”标签，就可以可视化页面的分层情况，如下图所示：



渲染引擎给页面多图层示意图

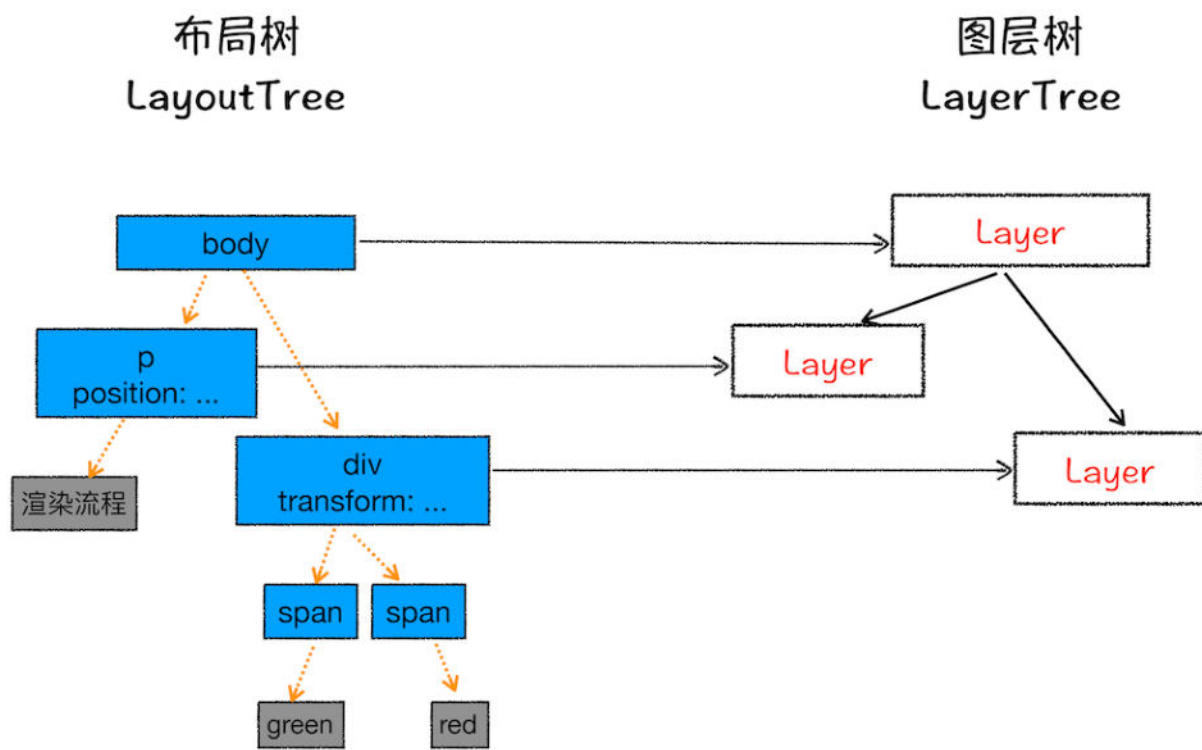
从上图可以看出，渲染引擎给页面分了很多图层，这些图层按照一定顺序叠加在一起，就形成了最终的页面，你可以参考下图：



图层叠加的最终展示页面

现在你知道了浏览器的页面实际上被分成了很多图层，这些图层叠加后合成了最终的页面。下面我们再来看看这些图层和布局树节点之间的关系，如文中图所示：





布局树和图层树关系示意图

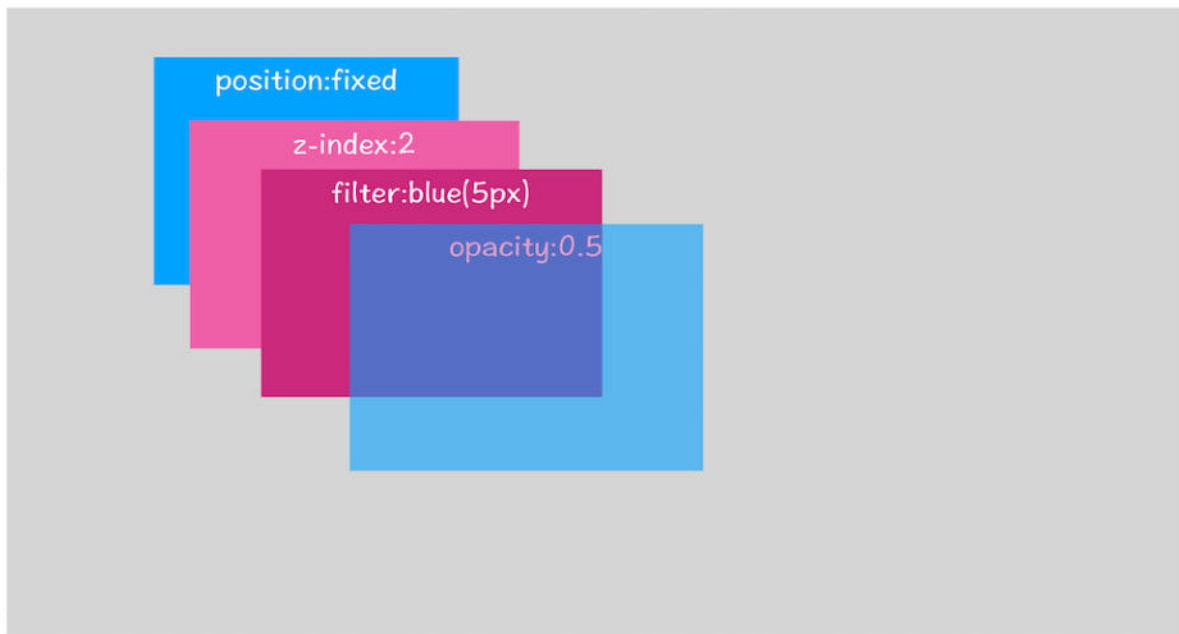
通常情况下，并不是布局树的每个节点都包含一个图层，如果一个节点没有对应的层，那么这个节点就从属于父节点的图层。如上图中的 `span` 标签没有专属图层，那么它们就从属于它们的父节点图层。但不管怎样，最终每一个节点都会直接或者间接地从属于一个层。

那么需要满足什么条件，渲染引擎才会为特定的节点创建新的图层呢？通常满足下面两点中任意一点的元素就可以被提升为单独的一个图层。

**第一点，拥有层叠上下文属性的元素会被提升为单独的一层。**

页面是个二维平面，但是层叠上下文能够让 HTML 元素具有三维概念，这些 HTML 元素按照自身属性的优先级分布在垂直于这个二维平面的  $z$  轴上。你可以结合下图来直观感受下：





层叠上下文示意图

从图中可以看出，明确定位属性的元素、定义透明属性的元素、使用 CSS 滤镜的元素等，都拥有层叠上下文属性。

若你想要了解更多层叠上下文的知识，你可以 [参考这篇文章](#)。

**第二点，需要剪裁（clip）的地方也会被创建为图层。**

不过首先你需要了解什么是剪裁，结合下面的 HTML 代码：

 复制代码

```
1 <style>
2     div {
3         width: 200;
4         height: 200;
5         overflow:auto;
6         background: gray;
7     }
8 </style>
9 <body>
10     <div >
```

```
11         <p>所以元素有了层叠上下文的属性或者需要被剪裁，那么就会被提升成为单独一层，你可以参看下
12         <p>从上图我们可以看到，document层上有A和B层，而B层之上又有两个图层。这些图层组织在一
13         <p>图层树是基于布局树来创建的，为了找出哪些元素需要在哪些层中，渲染引擎会遍历布局树来创
14     </div>
```



在这里我们把 div 的大小限定为 200 \* 200 像素，而 div 里面的文字内容比较多，文字所显示的区域肯定会超出 200 \* 200 的面积，这时候就产生了剪裁，渲染引擎会把裁剪文字内容的一部分用于显示在 div 区域，下图是运行时的执行结果：

属性或者需要被剪裁，那么就会被提升成为单独一层，你可以参看下图：

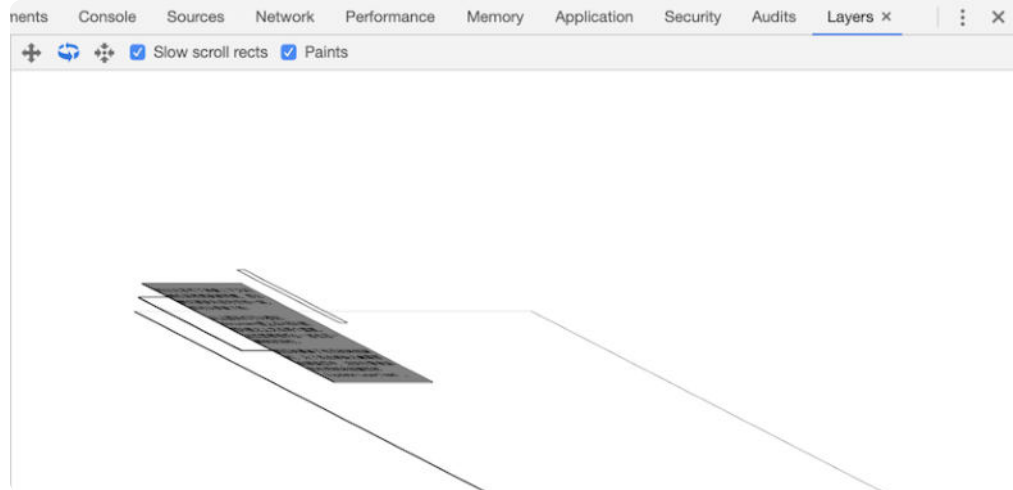
从上图我们可以看到，document层上有A和B层，而B层之上又有两个图层。这些图层组织在一起也是一颗树状结构。

#### 剪裁执行结果

出现这种裁剪情况的时候，渲染引擎会为文字部分单独创建一个层，如果出现滚动条，滚动条也会被提升为单独的层。你可以参考下图：







属性或者需要被剪裁，那么就会被提升成为单独一层，你可以参看下图：

从上图我们可以看到，document层上有A和B层，而B层之上又有两个图层。这些图层组织在一起也是一颗树状结构。

被裁剪的内容会出现在单独一层

所以说，元素有了层叠上下文的属性或者需要被剪裁，满足其中任意一点，就会被提升成为单独一层。

## 图层绘制

在完成图层树的构建之后，渲染引擎会对图层树中的每个图层进行绘制，那么接下来我们看看渲染引擎是怎么实现图层绘制的？

试想一下，如果给你一张纸，让你先把纸的背景涂成蓝色，然后在中间位置画一个红色的圆，最后再在圆上画个绿色三角形。你会怎么操作呢？

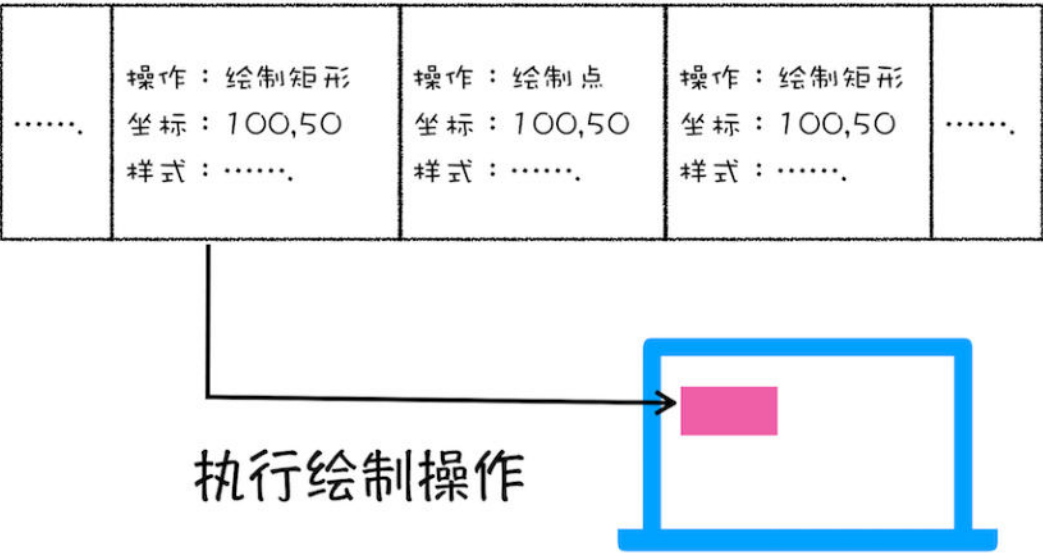
通常，你会把你的绘制操作分解为三步：

1. 绘制蓝色背景；
2. 在中间绘制一个红色的圆；
3. 再在圆上绘制绿色三角形。

渲染引擎实现图层的绘制与之类似，会把一个图层的绘制拆分成很多小的**绘制指令**，然后再把这些指令按照顺序组成一个待绘制列表，如下图所示：



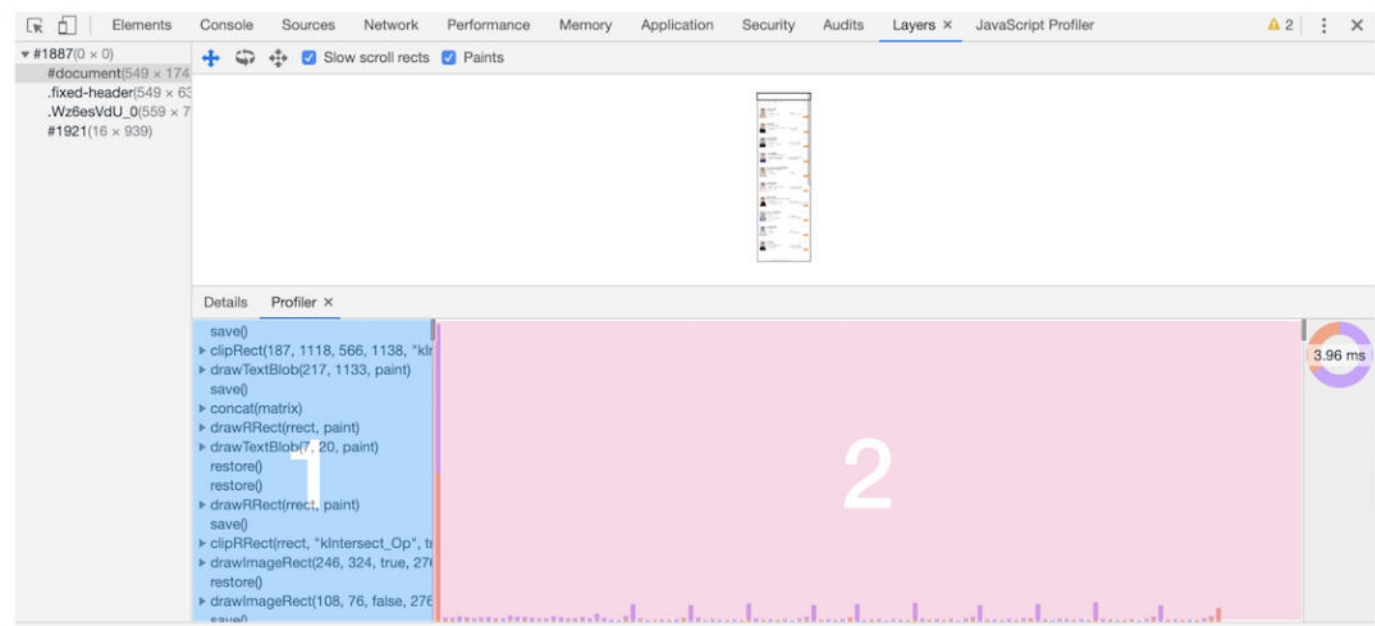
# 绘制列表



绘制列表

从图中可以看出，绘制列表中的指令其实非常简单，就是让其执行一个简单的绘制操作，比如绘制粉色矩形或者黑色的线等。而绘制一个元素通常需要好几条绘制指令，因为每个元素的背景、前景、边框都需要单独的指令去绘制。所以在图层绘制阶段，输出的内容就是这些待绘制列表。

你也可以打开“开发者工具”的“Layers”标签，选择“document”层，来实际体验下绘制列表，如下图所示：

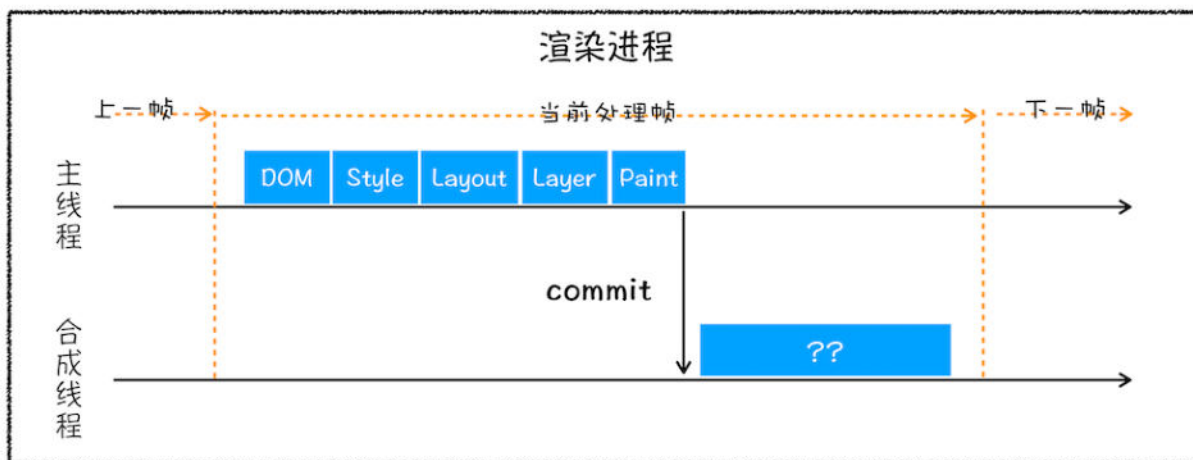




在该图中，区域 1 就是 document 的绘制列表，拖动区域 2 中的进度条可以重现列表的绘制过程。

## 栅格化（raster）操作

绘制列表只是用来记录绘制顺序和绘制指令的列表，而实际上绘制操作是由渲染引擎中的合成线程来完成的。你可以结合下图来看下渲染主线程和合成线程之间的关系：



渲染进程中的合成线程和主线程

如上图所示，当图层的绘制列表准备好之后，主线程会把该绘制列表**提交（commit）**给合成线程，那么接下来合成线程是怎么工作的呢？

那我们得先来看看什么是视口，你可以参看下图：





屏幕上页面的可见区域  
就叫视口 (ViewPort)

视口

通常一个页面可能很大，但是用户只能看到其中的一部分，我们把用户可以看到的这个部分叫做**视口** (viewport)。

在有些情况下，有的图层可以很大，比如有的页面你使用滚动条要滚动好久才能滚动到底部，但是通过视口，用户只能看到页面的很小一部分，所以在这种情况下，要绘制出所有图层内容的话，就会产生太大的开销，而且也没有必要。

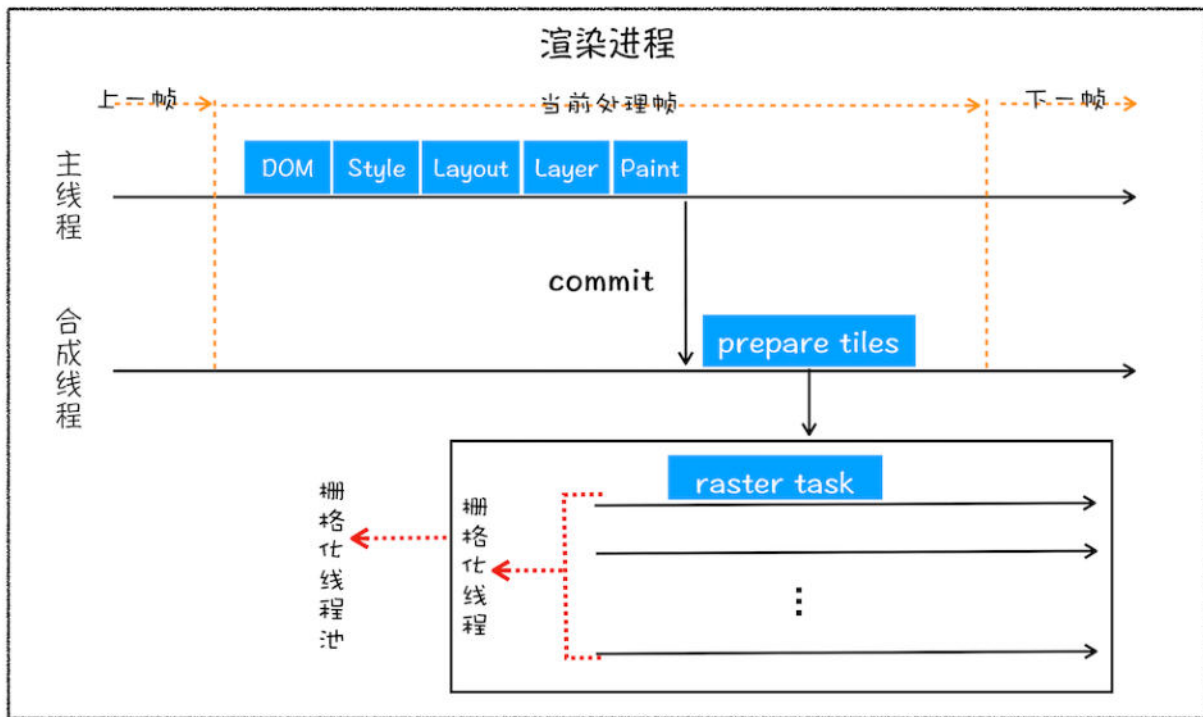
基于这个原因，**合成线程会将图层划分为图块 (tile)**，这些图块的大小通常是 256x256 或者 512x512，如下图所示：



图层被划分为图块示意图

然后合成线程会按照视口附近的图块来优先生成位图，实际生成位图的操作是由栅格化来执行的。所谓栅格化，是指将图块转换为位图。而图块是栅格化执行的最小单位。渲染进程维护了一个栅格化的线程池，所有的图块栅格化都是在线程池内执行的，运行方式如下图所示：



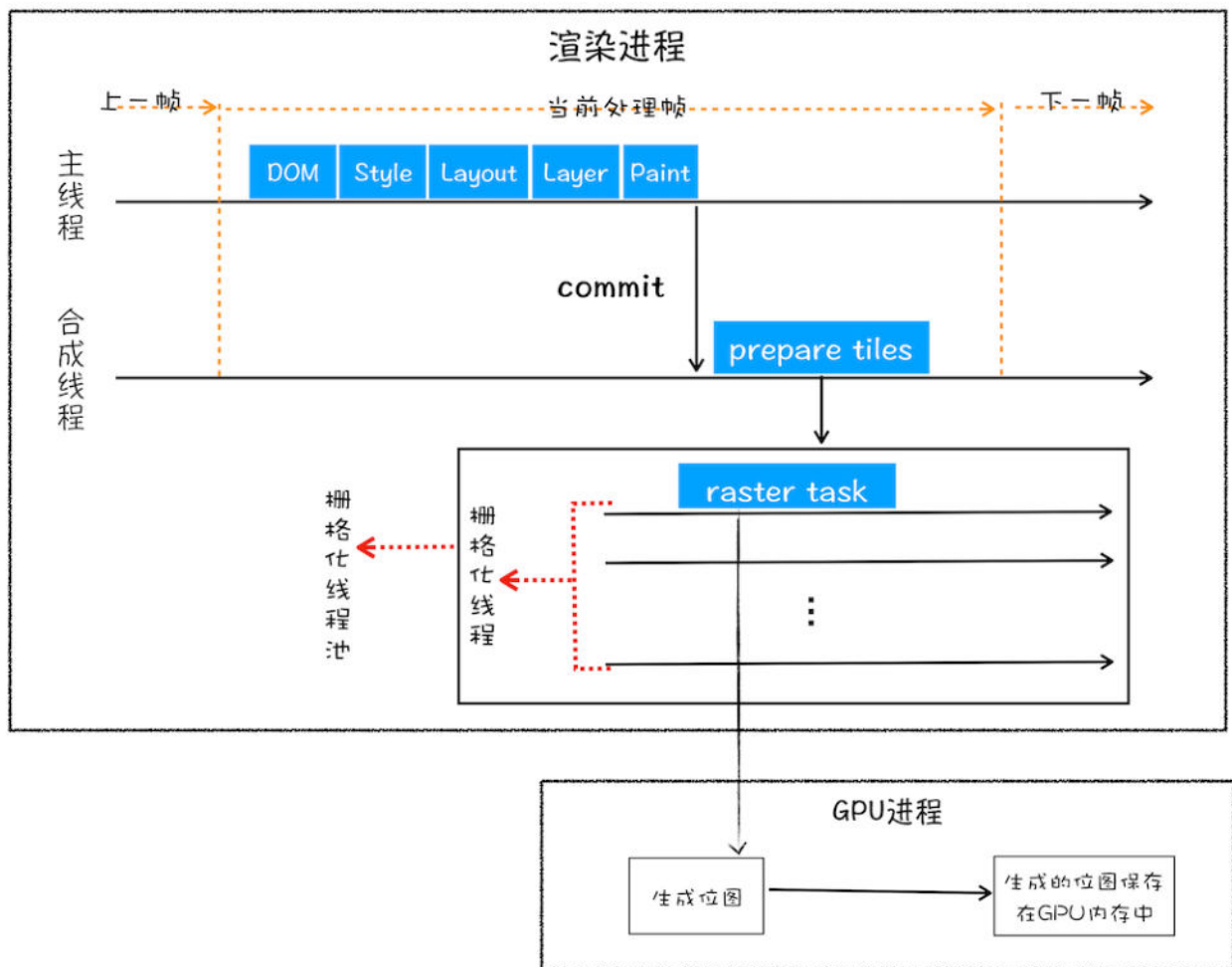


合成线程提交图块给栅格化线程池

通常，栅格化过程都会使用 GPU 来加速生成，使用 GPU 生成位图的过程叫快速栅格化，或者 GPU 栅格化，生成的位图被保存在 GPU 内存中。

相信你还记得，GPU 操作是运行在 GPU 进程中，如果栅格化操作使用了 GPU，那么最终生成位图的操作是在 GPU 中完成的，这就涉及到了跨进程操作。具体形式你可以参考下图：





GPU 栅格化

从图中可以看出，渲染进程把生成图块的指令发送给 GPU，然后在 GPU 中执行生成图块的位图，并保存在 GPU 的内存中。

## 合成和显示

一旦所有图块都被光栅化，合成线程就会生成一个绘制图块的命令——“DrawQuad”，然后将该命令提交给浏览器进程。

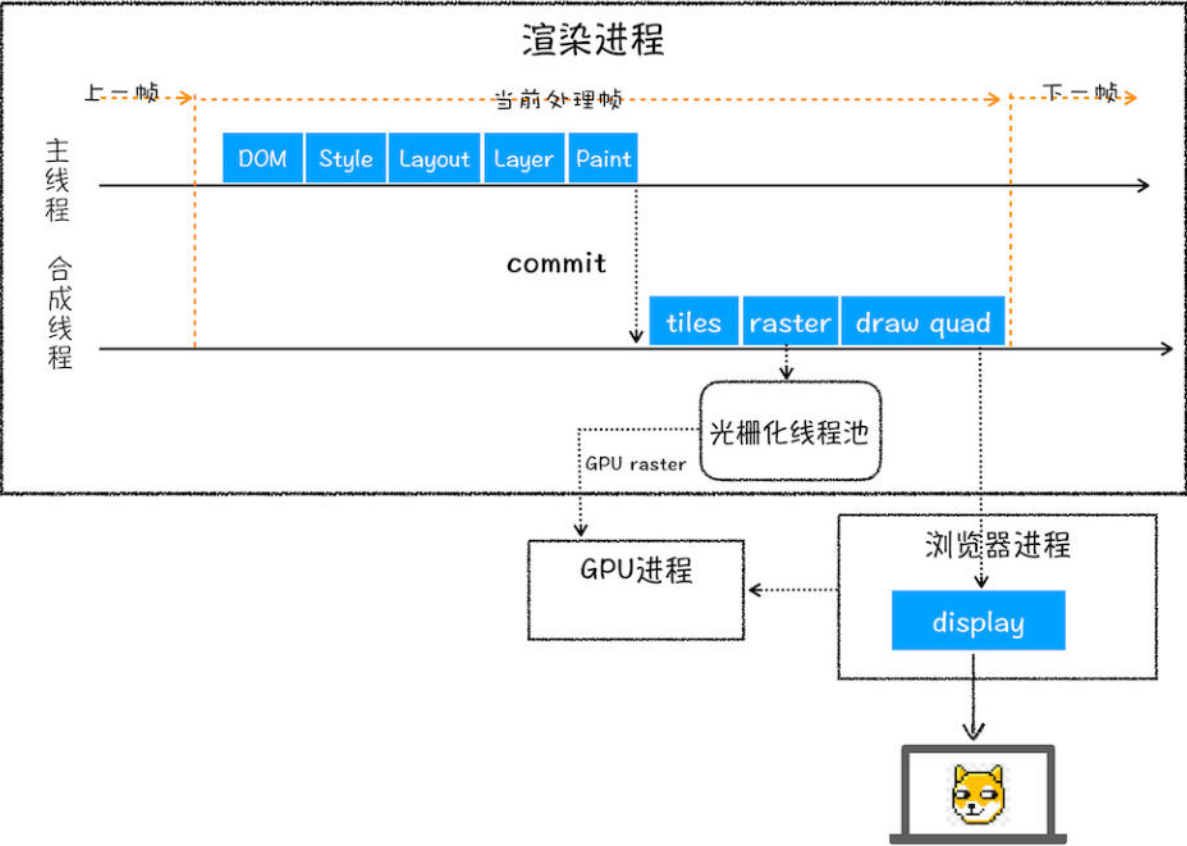
浏览器进程里面有一个叫 viz 的组件，用来接收合成线程发过来的 DrawQuad 命令，然后根据 DrawQuad 命令，将其页面内容绘制到内存中，最后再将内存显示在屏幕上。

到这里，经过一系列的阶段，编写好的 HTML、CSS、JavaScript 等文件，经过浏览器就会显示出漂亮的页面了。

## 渲染流水线大总结



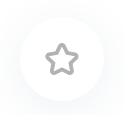
好了，我们现在已经分析完了整个渲染流程，从 HTML 到 DOM、样式计算、布局、图层、绘制、光栅化、合成和显示。下面我用一张图来总结下这整个渲染流程：



完整的渲染流水线示意图

结合上图，一个完整的渲染流程大致可总结为如下：

1. 渲染进程将 HTML 内容转换为能够读懂的 **DOM 树** 结构。
2. 渲染引擎将 CSS 样式表转化为浏览器可以理解的 **styleSheets**，计算出 DOM 节点的风格式。
3. 创建**布局树**，并计算元素的布局信息。
4. 对布局树进行分层，并生成**分层树**。
5. 为每个图层生成**绘制列表**，并将其提交到合成线程。
6. 合成线程将图层分成**图块**，并在**光栅化线程池**中将图块转换成位图。
7. 合成线程发送绘制图块命令 **DrawQuad** 给浏览器进程。
8. 浏览器进程根据 DrawQuad 消息**生成页面**，并**显示**到显示器上。



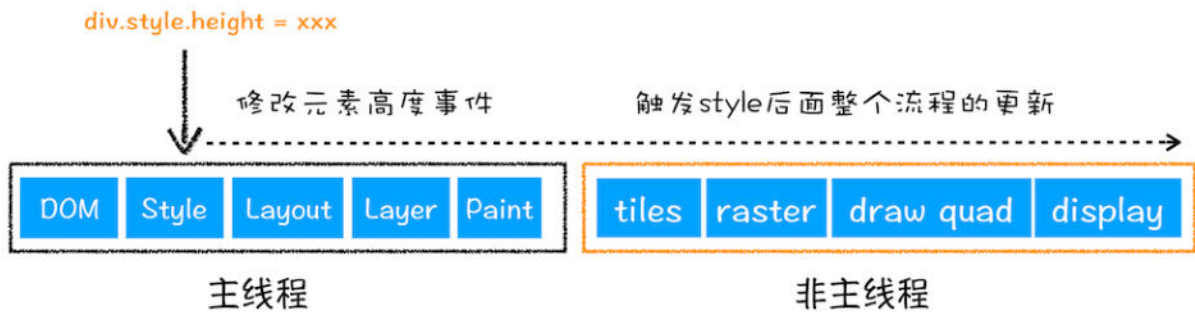


# 相关概念

有了上面介绍渲染流水线的基础，我们再来看看三个和渲染流水线相关的概念——“重排”“重绘”和“合成”。理解了这三个概念对于你后续 Web 的性能优化会有很大帮助。

## 1. 更新了元素的几何属性（重排）

你可先参考下图：

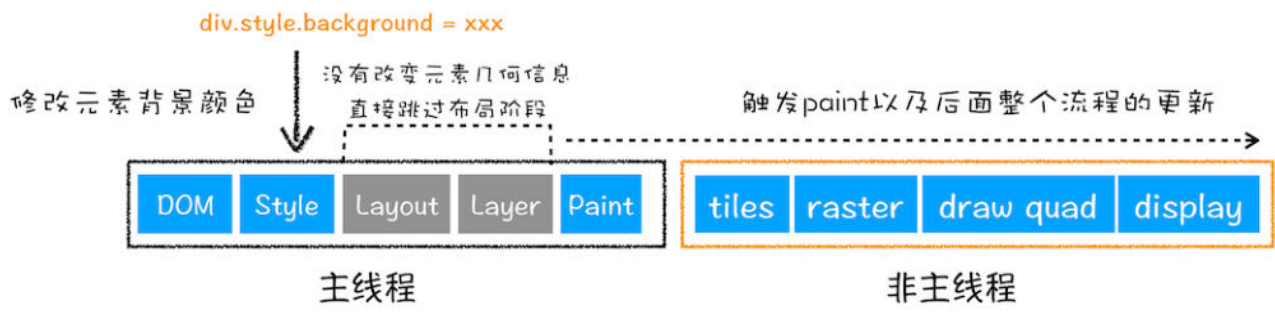


更新元素的几何属性

从上图可以看出，如果你通过 JavaScript 或者 CSS 修改元素的几何位置属性，例如改变元素的宽度、高度等，那么浏览器会触发重新布局，解析之后的一系列子阶段，这个过程就叫重排。无疑，重排需要更新完整的渲染流水线，所以开销也是最大的。

## 2. 更新元素的绘制属性（重绘）

接下来，我们再来看看重绘，比如通过 JavaScript 更改某些元素的背景颜色，渲染流水线会怎样调整呢？你可以参考下图：



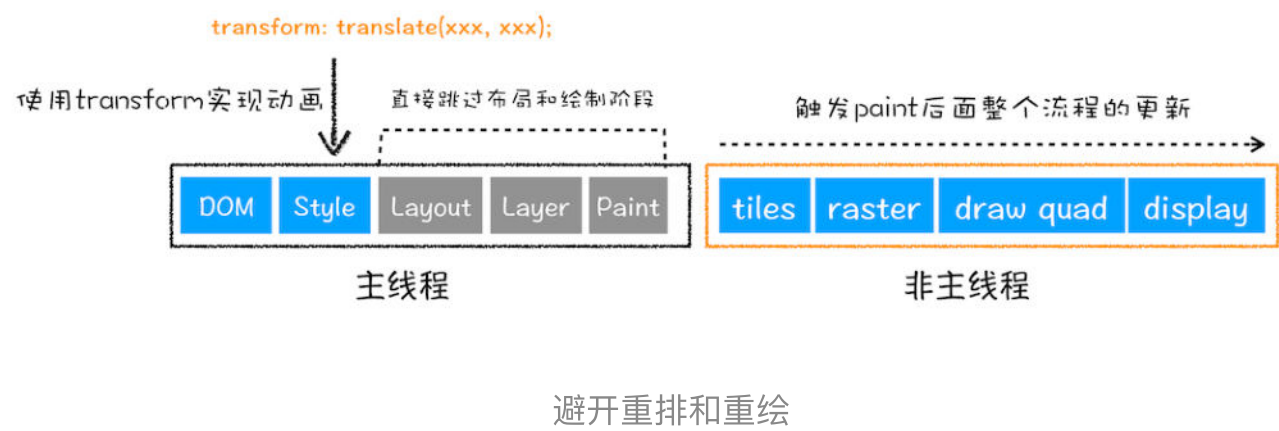
更新元素背景



从图中可以看出，如果修改了元素的背景颜色，那么布局阶段将不会被执行，因为并没有引起几何位置的变换，所以就直接进入了绘制阶段，然后执行之后的一系列子阶段，这个过程就叫**重绘**。相较于重排操作，**重绘省去了布局 and 分层阶段，所以执行效率会比重排操作要高一些。**

### 3. 直接合成阶段

那如果你更改一个既不要布局也不要绘制的属性，会发生什么变化呢？渲染引擎将跳过布局和绘制，只执行后续的合成操作，我们把这个过程叫做**合成**。具体流程参考下图：



在上图中，我们使用了 CSS 的 transform 来实现动画效果，这可以避开重排和重绘阶段，直接在非主线程上执行合成动画操作。这样的效率是最高的，因为是在非主线程上合成，并没有占用主线程的资源，另外也避开了布局和绘制两个子阶段，所以**相对于重绘和重排，合成能大大提升绘制效率。**

至于如何用这些概念去优化页面，我们会在后面相关章节做详细讲解的，这里你只需要先结合“渲染流水线”弄明白这三个概念及原理就行。

### 总结

通过本文的分析，你应该可以看到，Chrome 的渲染流水线还是相当复杂晦涩，且难以理解，不过 Chrome 团队在不断添加新功能的同时，也在不断地重构一些子阶段，目的就是**让整体渲染架构变得更加简单和高效**，正所谓大道至简。

通过这么多年的生活和工作经验来看，无论是做架构设计、产品设计，还是具体到代码的实现，甚至处理生活中的一些事情，能够把复杂问题简单化的人都是具有大智慧的。所以，在工作或生活中，你若想要简化遇到的问题，就要刻意地练习，练就抓住问题本质的能力，把那些复杂的问题简单化，从而最终真正解决问题。




## 思考时间

在优化 Web 性能的方法中，减少重绘、重排是一种很好的优化方式，那么结合文中的分析，你能总结出来为什么减少重绘、重排能优化 Web 性能吗？那又有那些具体的实践方法能减少重绘、重排呢？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

分享给需要的人，Ta订阅超级会员，你将得 50 元

Ta单独购买本课程，你将得 20 元

 生成海报并分享

 赞 61

 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 05 | 渲染流程（上）：HTML、CSS和JavaScript，是如何变成页面的？

下一篇 07 | 变量提升：JavaScript代码是按顺序执行的吗？



# JVM + NIO + Spring

各大厂面试题及知识点详解

限时免费



## 精选留言 (124)

写留言



Hurry

2019-08-18

减少重排重绘, 方法很多:

1. 使用 class 操作样式, 而不是频繁操作 style
2. 避免使用 table 布局
3. 批量dom 操作, 例如 createDocumentFragment, 或者使用框架, 例如 React
4. Debounce window resize 事件
5. 对 dom 属性的读写要分离
6. will-change: transform 做优化

作者回复: 总结的很好, 很有经验👍

共 15 条评论 >

155



Zkerhcy

2020-03-16

浏览器工作流程『从输入 URL 到页面展示』学习笔记

导航

用户输入



1. 用户在地址栏按下回车，检查输入（关键字 or 符合 URL 规则），组装完整 URL；
2. 回车前，当前页面执行 onbeforeunload 事件；
3. 浏览器进入加载状态。

## URL 请求

1. 浏览器进程通过 IPC 把 URL 请求发送至网络进程；
2. 查找资源缓存（有效期内）；
3. DNS 解析（查询 DNS 缓存）；
4. 进入 TCP 队列（单个域名 TCP 连接数量限制）；
5. 创建 TCP 连接（三次握手）；
6. HTTPS 建立 TLS 连接（client hello, server hello, pre-master key 生成『对话密钥』）；
7. 发送 HTTP 请求（请求行[方法、URL、协议]、请求头 Cookie 等、请求体 POST）；
8. 接受请求（响应行[协议、状态码、状态消息]、响应头、响应体等）；
  - 状态码 301 / 302，根据响应头中的 Location 重定向；
  - 状态码 200，根据响应头中的 Content-Type 决定如何响应（下载文件、加载资源、渲染 HTML）。

## 准备渲染进程

1. 根据是否同一站点（相同的协议和根域名），决定是否复用渲染进程。

## 提交文档

1. 浏览器进程接受到网路进程的响应头数据，向渲染进程发送『提交文档』消息；
2. 渲染进程收到『提交文档』消息后，与网络进程建立传输数据『管道』；
3. 传输完成后，渲染进程返回『确认提交』消息给浏览器进程；
4. 浏览器接受『确认提交』消息后，移除旧文档、更新界面、地址栏，导航历史状态等；
5. 此时标识浏览器加载状态的小圆圈，从此前 URL 网络请求时的逆时针选择，即将变成顺时针旋转（进入渲染阶段）。

## 渲染

### 渲染流水线

### 构建 DOM 树

1. 输入：HTML 文档；
2. 处理：HTML 解析器解析；



3. 输出：DOM 数据解构。

## 样式计算

1. 输入：CSS 文本；
2. 处理：属性值标准化，每个节点具体样式（继承、层叠）；
3. 输出：styleSheets(CSSOM)。

## 布局(DOM 树中元素的计划位置)

1. DOM & CSSOM 合并成渲染树；
2. 布局树（DOM 树中的可见元素）；
3. 布局计算。

## 分层

1. 特定节点生成专用图层，生成一棵图层树（层叠上下文、Clip，类似 PhotoShop 里的图层）；
2. 拥有层叠上下文属性（明确定位属性、透明属性、CSS 滤镜、z-index 等）的元素会创建单独图层；
3. 没有图层的 DOM 节点属于父节点图层；
4. 需要剪裁的地方也会创建图层。

## 绘制指令

1. 输入：图层树；
2. 渲染引擎对图层树中每个图层进行绘制；
3. 拆分成绘制指令，生成绘制列表，提交到合成线程；
4. 输出：绘制列表。

## 分块

1. 合成线程会将较大、较长的图层（一屏显示不完，大部分不在视口内）划分为图块（tile, 256\*256, 512\*512）。

## 光栅化（栅格化）

1. 在光栅化线程池中，将视口附近的图块优先生成位图（栅格化执行该操作）；
2. 快速栅格化：GPU 加速，生成位图（GPU 进程）。

## 合成绘制





1. 绘制图块命令——DrawQuad，提交给浏览器进程；
2. 浏览器进程的 viz 组件，根据DrawQuad命令，绘制在屏幕上。

## 相关概念

### 重排

1. 更新了元素的几何属性（如宽、高、边距）；
2. 触发重新布局，解析之后的一系列子阶段；
3. 更新完成的渲染流水线，开销最大。

### 重绘

1. 更新元素的绘制属性（元素的颜色、背景色、边框等）；
2. 布局阶段不会执行（无几何位置变换），直接进入绘制阶段。

### 合成

1. 直接进入合成阶段（例如CSS的 transform 动画）；
2. 直接执行合成阶段，开销最小。

共 2 条评论 >

👍 150



Luke

2019-09-27

关于浏览器渲染的知识点讲的很细致，我想问下，关于浏览器的渲染细节的知识老师是从哪里学到的？，是通过研究源码学习的吗？有没有一些好的学习资料或者学习方法推荐？能否专门出一篇“授人以渔”的文章，谢谢！

作者回复: 主要几个途径：

1:chromium源码

2:chromium源码里面的一些注释和文档

3:还有油管上blinkon上有一些深入讲解内核的视频

目前基本没有系统介绍浏览器知识的文档，而且网上很多文档还是比较早期的，很多内容都不太适合新版的浏览器了。

这里将浏览器知识和前端系统下结合起来是一件工作量非常大的事。



👍 88





明

2019-08-17

渲染进程里的帧的概念是什么样子的呢？一个page是一帧吗

作者回复: 可以拿放电影电影来解释，通常，电影的帧速是24，也就是说每秒切换24幅画面，其中的每幅画面就是一帧。

理解什么是帧后，我们在回过头看看我们的页面。由于目前大多数设备的屏幕刷新率为 60 次/秒。因此，如果页面中有一个动画、或一个渐变效果、或者用户正在滚动页面，那么浏览器渲染动画的频率至少要和刷新频率保持一致，也就是每秒需要更新60次，这样我们就能计算出来生成每帧的预算只有 (1/60) 毫秒，也就是16毫秒多一点(1 秒/ 60 = 16.66 毫秒)。如果超过16毫秒，帧率将下降，并且会出现画面抖动现象，此现象通常被称为卡顿，会对用户体验产生负面影响。

所以，如果想要保证画面的流畅，就需要尽量降低每帧的渲染时间，所以局部更新流水线显得非常重要了，能大大减少处理每帧所消耗的时间。

共 6 条评论 >

👍 53



mfist

2019-08-17

减少重排重绘，相当于少了渲染进程的主线程和非主线程的很多计算和操作，能够加快web的展示。

1 触发repaint reflow的操作尽量放在一起，比如改变dom高度和设置margin分开写，可能会出发两次重排

2 通过虚拟dom层计算出操作总得差异，一起提交给浏览器。之前还用过createdocumentfragment来汇总append的dom,来减少触发重排重绘次数。

作者回复: 很赞

共 3 条评论 >

👍 37



Geek\_East

2019-11-28

渲染流程的最后，应该是浏览器进程将Compositor Frame发送到GPU, GPU进行显示吧？

作者回复: 这块我没深入将了，因为结构比较复杂，chromium团队还在重构大的架构，既然你问到了，我就简要介绍下：



1:首先渲染进程里执行图层合成(Layer Compositor)，也就是生成图层的操作，具体地讲，渲染进程的合成线程接收到图层的绘制消息时，会通过光栅化线程池将其提交给GPU进程，在GPU进程中执行光栅化操作，执行完成，再将结果返回给渲染进程的合成线程，执行合成图层操作！

2:合成的图层会被提交给浏览器进程，浏览器进程里会执行显示合成(Display Compositor)，也就是将所有的图层合成为可以显示的页面图片。最终显示器显示的就是浏览器进程中合成的页面图片

共 3 条评论 >

👍 21



ytd

2019-08-17

请教下老师，canvas的渲染流程是什么样的呢？它不涉及dom，也就不涉及dom树、样式计算、布局、分层，canvas的绘制过程也是在渲染进程中进行的吗？

作者回复: canvas绘制流程很简单，就是调用api直接在画布上绘制，没有DOM，也没有太多套路！

所有的绘制都是自己程序控制的

共 6 条评论 >

👍 16



杨陆伟

2019-08-17

最后的一段话非常经典，赞！大道至简，这真是做软件该秉持的原则，如果实现功能时感受到复杂和无序，那一定是那里错了

作者回复: 说的好，如果感觉到复杂和无序，那一定是哪里错了



👍 16



tokey

2019-08-17

老师您好！

我想问以下两个问题：

问题1：手机端开发，body 被内容撑开了，超过一屏，在滑动的过程中会不会触发重排，为什么？

问题2：如果 body 高度设置了100%

作者回复: 现代浏览器做了优化，把滚动操作交给了合成线程来处理，也就是说滚动的内容会被当成一个单独的图层，发生滚动的事件的时候，图层直接由合成线程来生成，也就是说这种情况下没有占用主线程，所以通常情况下不会产生重排和重回操作，只是简单合成就可以了，这样效率是最高的！



为什么说“通常”呢？这是因为目前渲染流程还是很复杂的，在滚动页面时，有些情况下，如果合成线程搞不定的，那么还要交给主线程去处理，这时候就涉及到重拍了，不过技术是往前发展的，渲染流程会变得越来越简单高效！

共 4 条评论 >

👍 11



番茄

2019-10-14

最后一部分，合成和显示讲的太模糊的，不是很理解。

共 1 条评论 >

👍 9



splm

2019-10-12

在GPU进程完成栅格化，并把结果保存在GPU内存中，此时的结果仍然保存在独立进程中。那么从渲染进程的合成线程发送Drawquad命令到浏览器主线程调用Viz组件，主进程是在什么时候拿到之前存在GPU内存中的位图结果的？是Viz主动去GPU内存获取这部分结果进行合成的吗？这里没太看懂。

共 3 条评论 >

👍 8



不存在的

2019-10-20

什么叫既不要布局也不要绘制的属性呢？

作者回复: 比如CSS3的实现的一些动画效果



👍 8



Luke

2019-09-10

老师，你好，我有几个问题一直都很很困惑，也没找到答案，希望老师能解惑一下，感谢！

1、图层、图块与BFC有什么区别联系吗？为什么BFC内元素的变动不会对BFC外的元素产生任何影响？是因为BFC会产生一个独立的图层或图块，渲染的时候只用重新渲染这一个图层或图块吗？BFC的原理是什么？

2、在划分图层的时候，每个图层都会生成一系列的绘制指令，而在划分图块的时候，一个图块可能包含多个图层，一个图层也可能分成多个图块，那么在将图块绘制成位图的时候，是如何执行绘制指令的？需要将绘制指令再划分到不同的图块中吗？

共 1 条评论 >

👍 7



悬炫

2019-08-26

老师文中说需要剪裁 (clip) 的地方也会被创建为图层，但是我复制了老师的代码后，发现需要剪裁的地方并没有单独的被创建为图层，难道是最新版本的谷歌浏览器改了渲染规则？  
我的浏览器版本是 76.0.3809.100 (正式版本) (64 位)

共 7 条评论 >

👍 5



**Fred 鱼**

2020-06-20

对于使用transform的元素，要事先定义好will-change:transform;，才能避免layout 和paint。



👍 4



**Warrior**

2019-09-20

重排是否只在当前分层中，会不会影响其他分层的重排？

作者回复: 目前会的，不过未来应该会解决这个问题



👍 5



**帅气小熊猫**

2019-08-19

这里的合成线程属于哪个进程？浏览器进程是指主进程吗？前面进程线程那块没有啊

作者回复: 合成线程属于渲染进程，你可以看文中示意图！

浏览器进程是主进程，负责提供一些基础服务和调度其它进程，你可以回顾下第一节和第四节内容。



👍 4



**板栗**

2019-09-18

老师，是将所有图块都栅格化，还是刚开始栅格化只可视区的图块，滚动的时候再去动态的栅格化。

共 5 条评论 >

👍 3



**无名**

2019-08-27

为何我的Layers标签中，选择了document，只有Details tab，没有Profile Tab？

共 2 条评论 >

👍 3





frankh

2019-08-21

transform为什么可以避免重排和重绘啊

作者回复: 因为直接在合成线程上执行就行了

共 2 条评论 >

 3

