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

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

现在我们有了布局树，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。

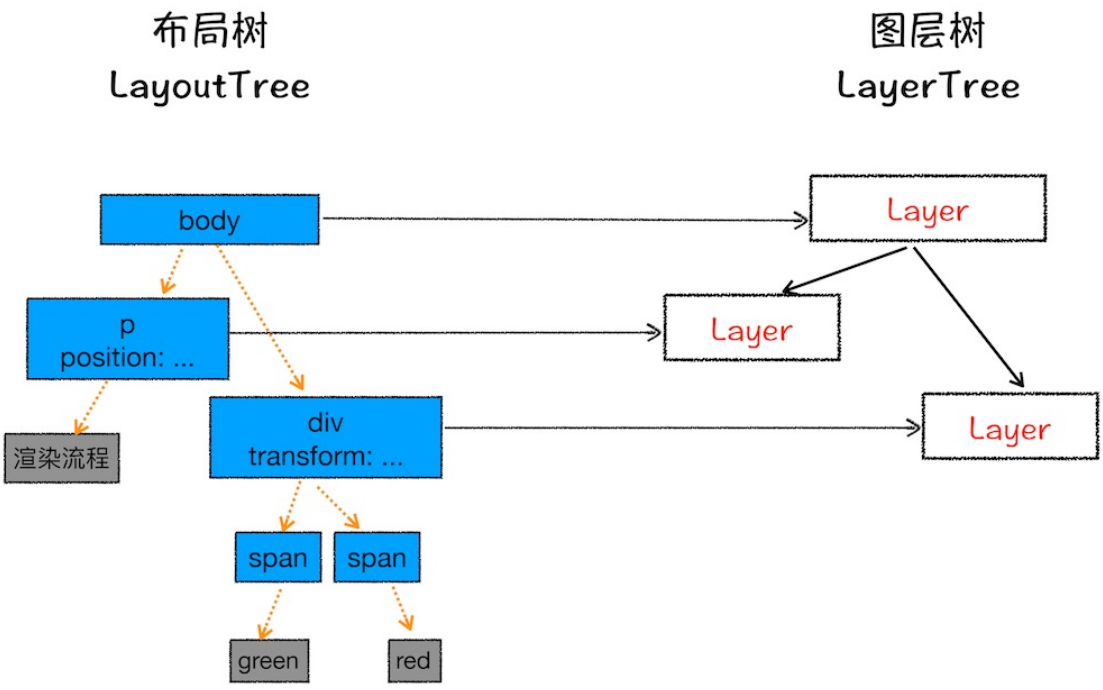
要想直观地理解什么是图层，你可以打开Chrome的“开发者工具”，选择“Layers”标签，就可以可视化页面的分层情况，如下图所示：

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：



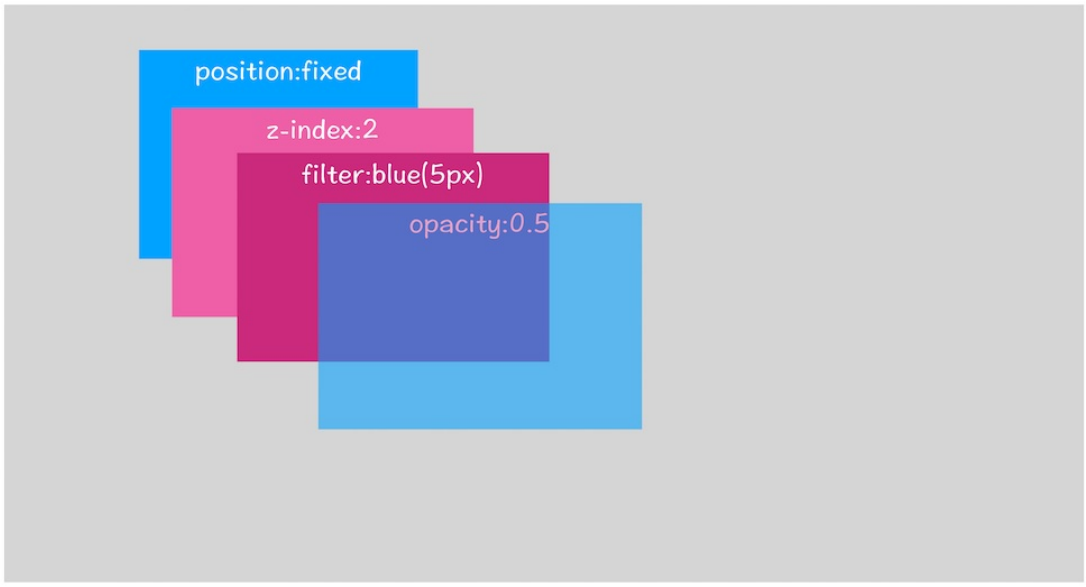
布局树和图层树关系示意图

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。

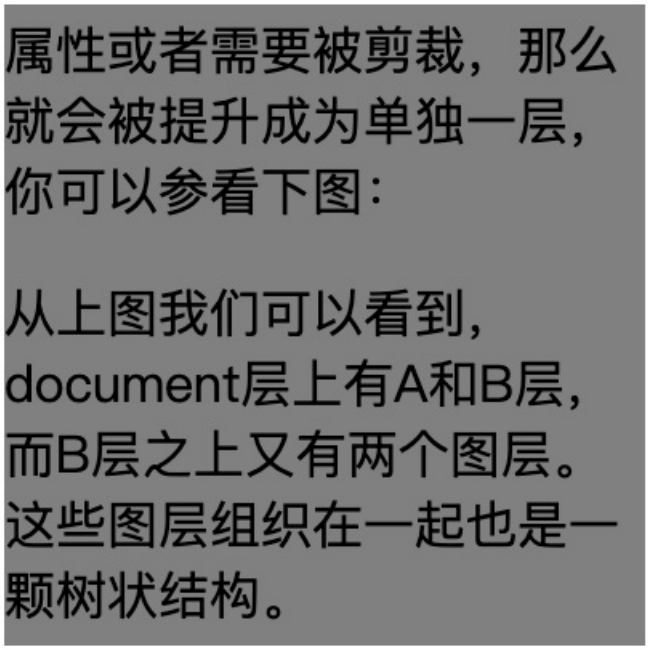
第二点，需要剪裁（clip）的地方也会被创建为图层。

不过首先你需要了解什么是剪裁，结合下面的HTML代码：

```
<style>
      div {
            width: 200;
            height: 200;
            overflow:auto;
            background: gray;
        }
</style>
<body>
```

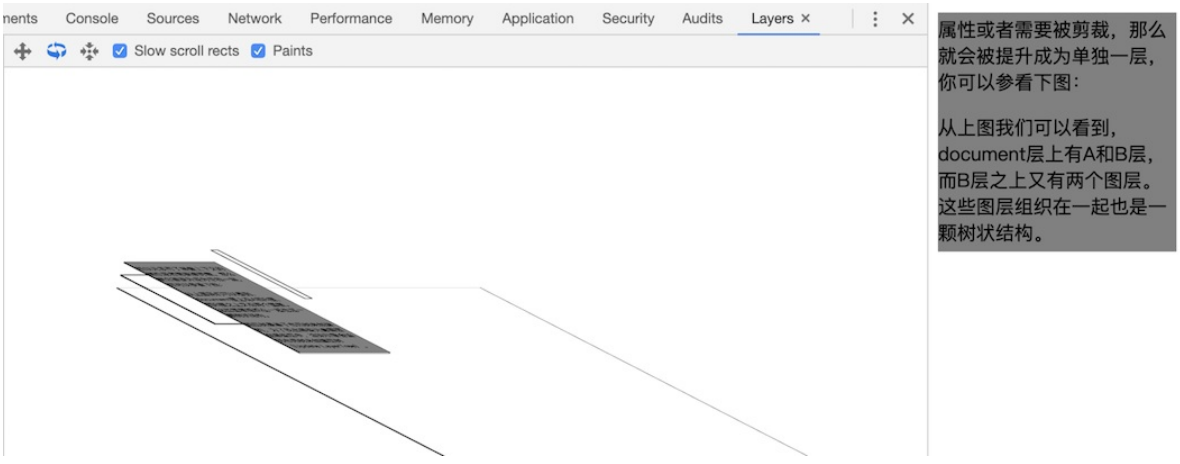
```
    <div >
        <p>所以元素有了层叠上下文的属性或者需要被剪裁，那么就会被提升成为单独一层，你可以参看下图：</p>
        <p>从上图我们可以看到，document层上有A和B层，而B层之上又有两个图层。这些图层组织在一起也是一颗树状结构。</p>
        <p>图层树是基于布局树来创建的，为了找出哪些元素需要在哪些层中，渲染引擎会遍历布局树来创建层树（Update LayerTree）。
    </div>
</body>
```

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：



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：



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，满足这任意一点，就会被提升成为单独一层。

图层绘制

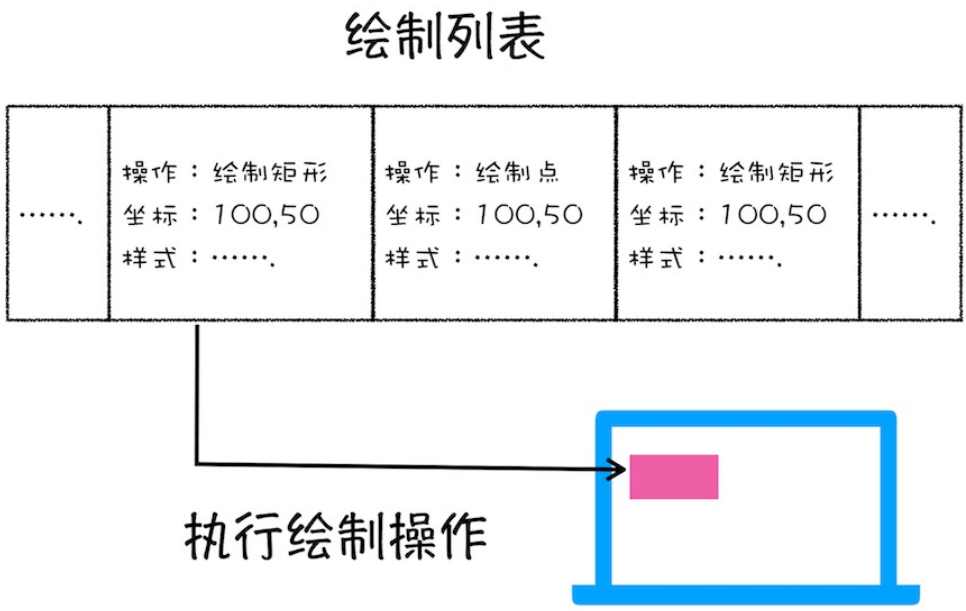
在完成图层树的构建之后，渲染引擎会对图层树中的每个图层进行绘制，那么接下来我们看看渲染引擎是怎么实现图层绘制的？

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。

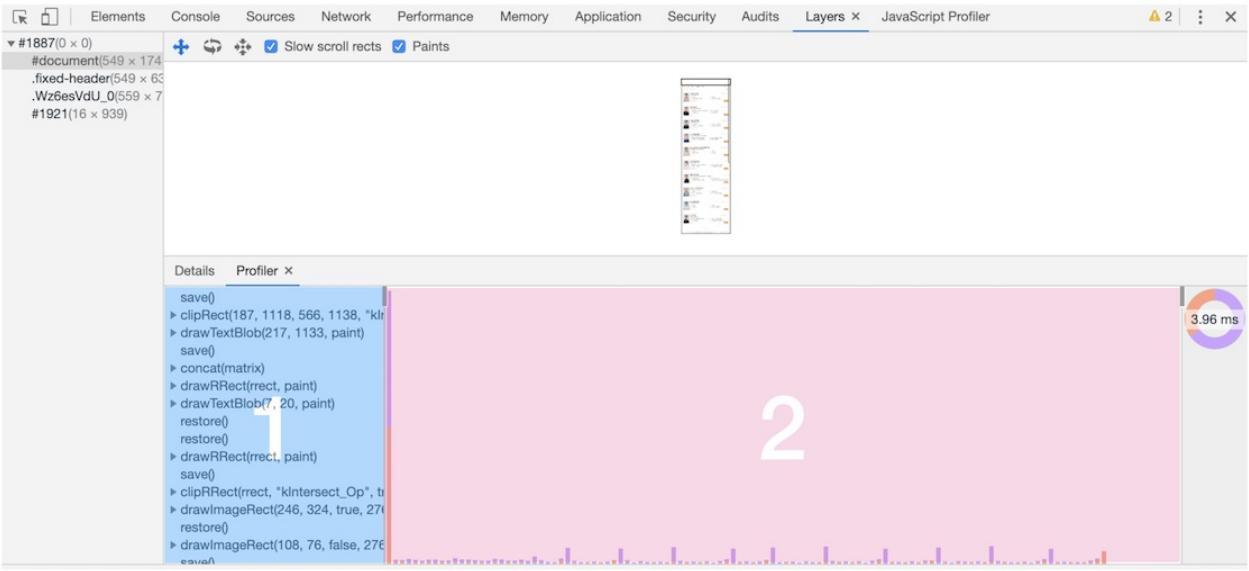
渲染引擎实现图层的绘制与之类似，会把一个图层的绘制拆分成很多小的**绘制指令**，然后再把这些指令按照顺序组成一个待绘制列表，如下图所示：



绘制列表

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：

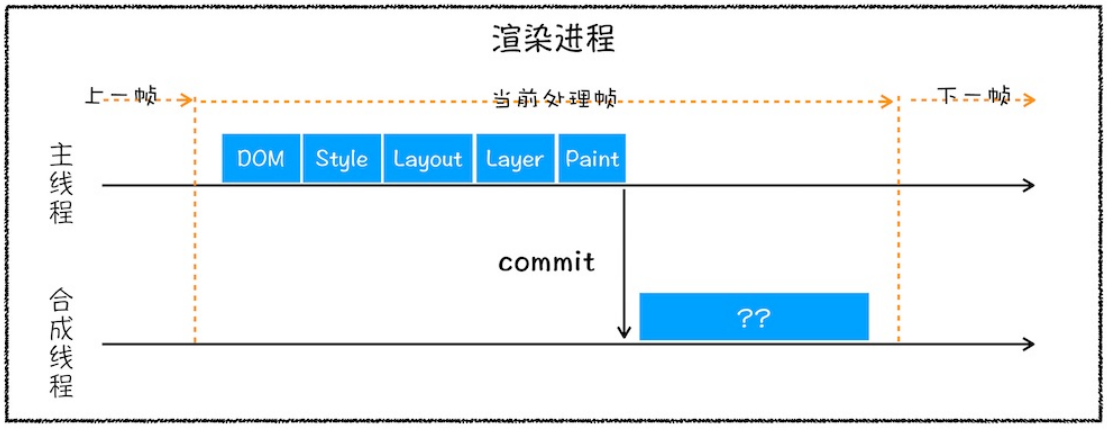


一个图层的绘制列表

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

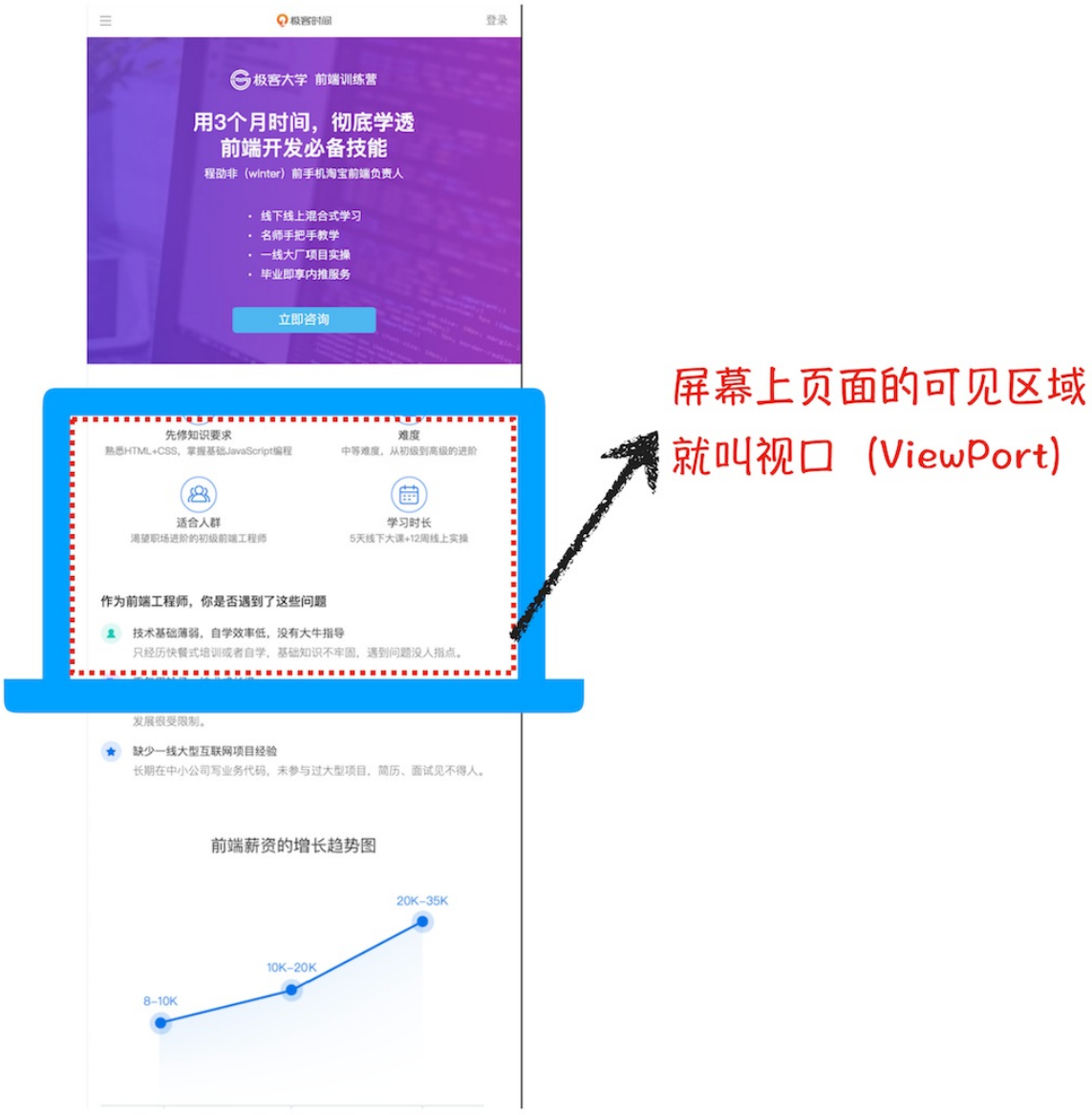
绘制列表只是用来记录绘制顺序和绘制指令的列表，而实际上绘制操作是由渲染引擎中的合成线程来完成的。你可以结合下图来看下渲染主线程和合成线程之间的关系：



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：

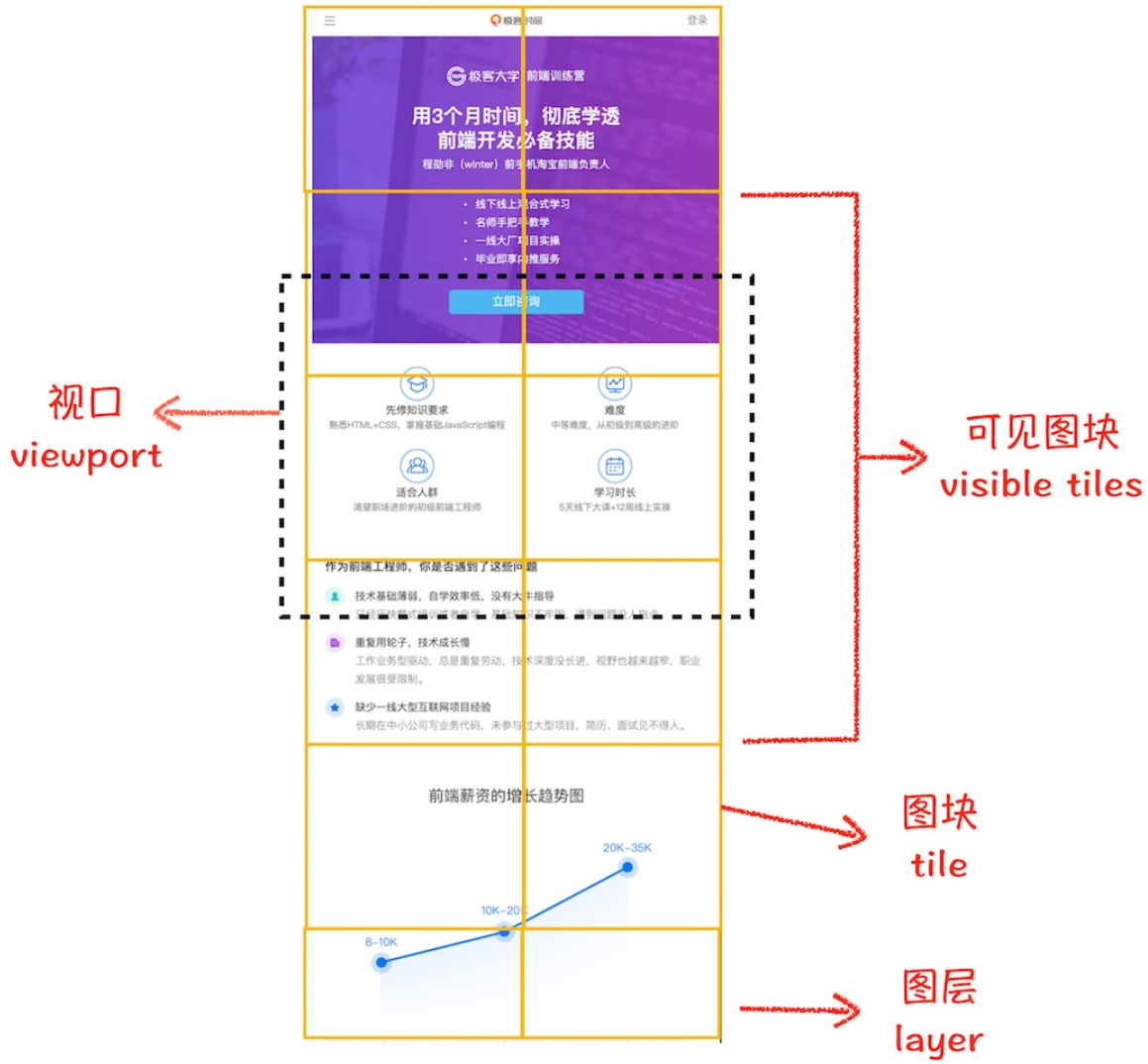


视口

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）。

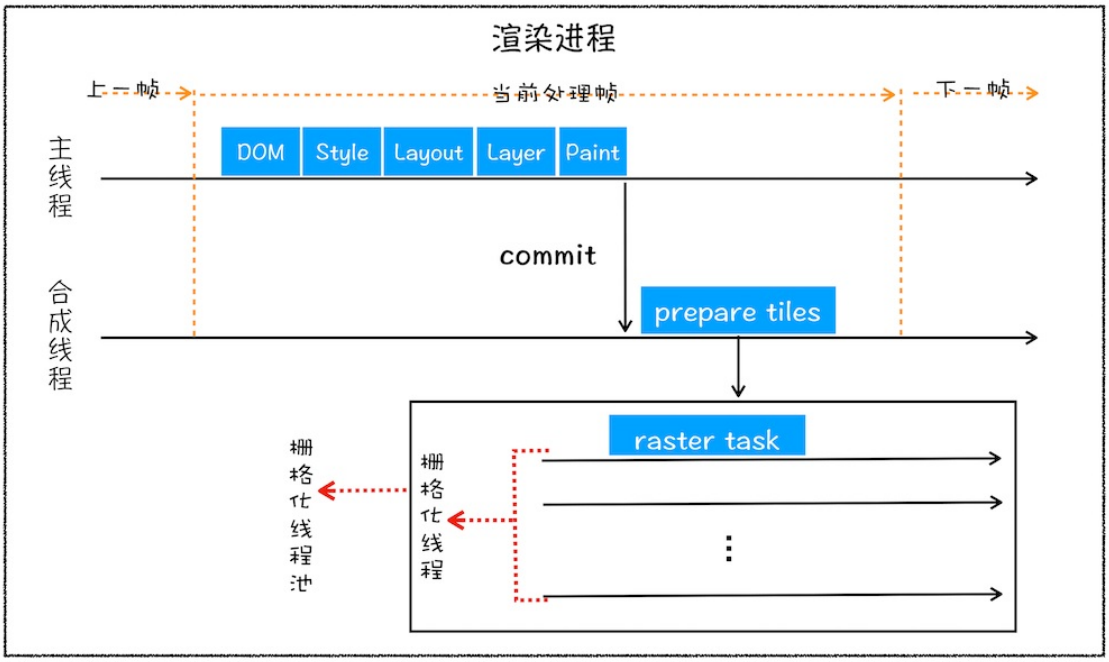
在有些情况下，有的图层可以很大，比如有的页面你使用滚动条要滚动好久才能滚动到底部，但是通过视口，用户只能看到页面的很小一部分，所以在这种情况下，要绘制出所有图层内容的话，就会产生太大的开销，而且也没有必要。

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

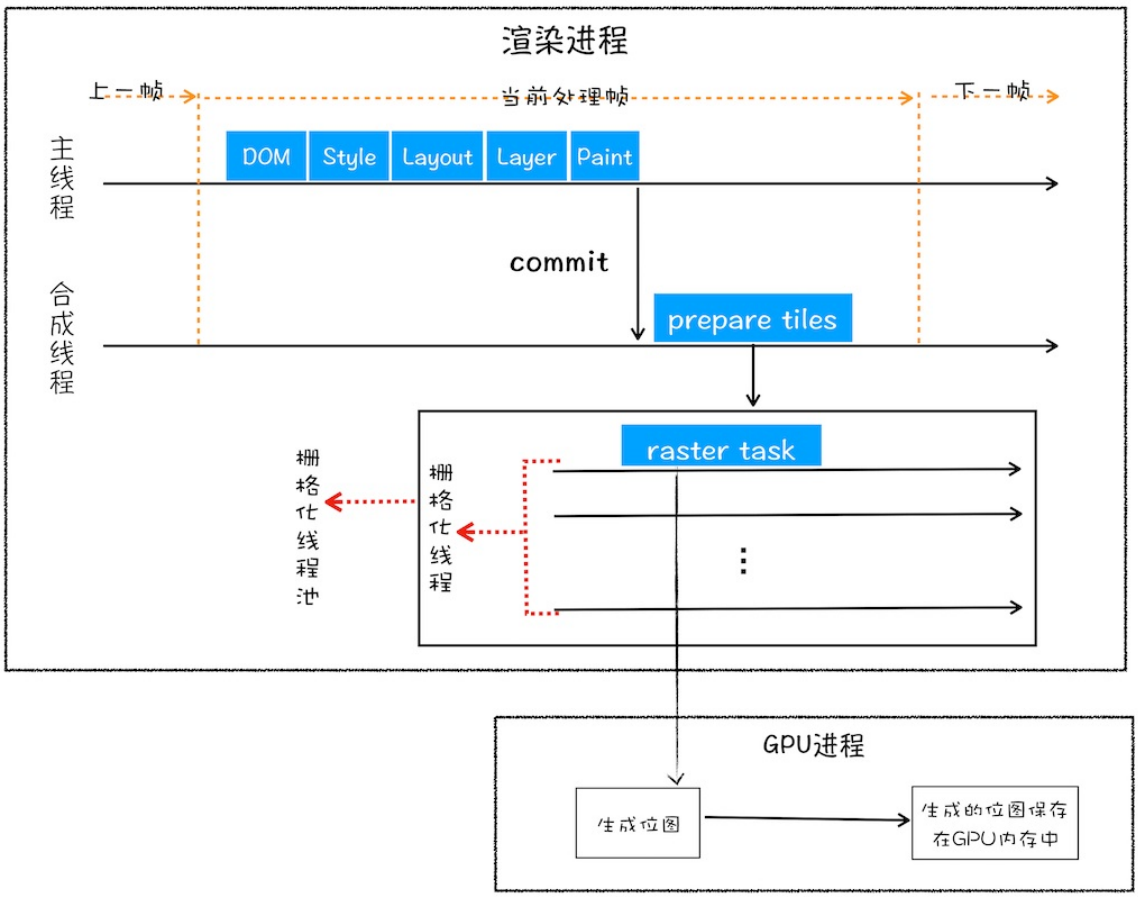
然后**合成线程会按照视口附近的图块来优先生成位图，实际生成位图的操作是由栅格化来执行的。所谓栅格化，是指将图块转换为位图**。而图块是栅格化执行的最小单位。渲染进程维护了一个栅格化的线程池，所有的图块栅格化都是在线程池内执行的，运行方式如下图所示：



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：



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

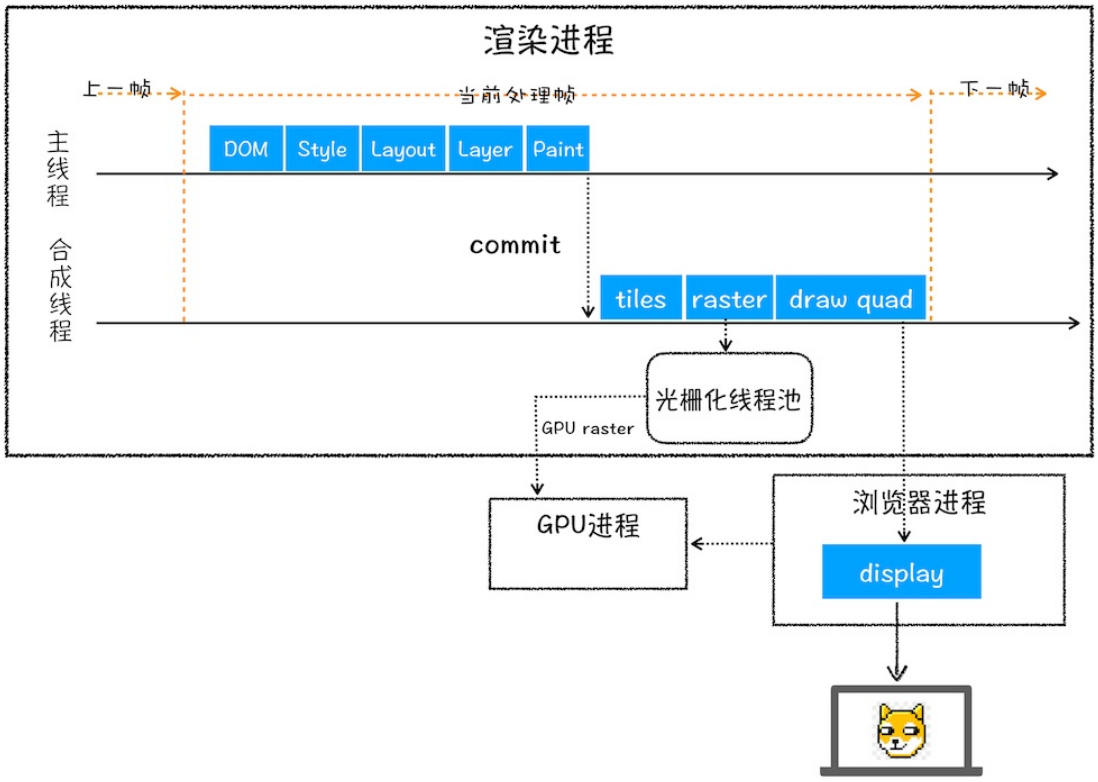
一旦所有图块都被光栅化，合成线程就会生成一个绘制图块的命令——“DrawQuad”，然后将该命令提交给浏览器进程。

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，经过这一系列的阶段，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节点的样式。
3. 创建**布局树**，并计算元素的布局信息。

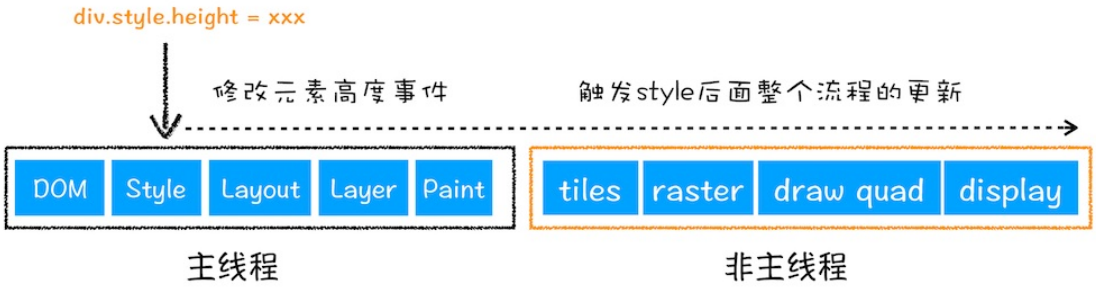
4. 对布局树进行分层，并生成**分层树**。
5. 为每个图层生成**绘制列表**，并将其提交到合成线程。
6. 合成线程将图层分成**图块**，并在**光栅化线程池**中将图块转换成位图。
7. 合成线程发送绘制图块命令**DrawQuad**给浏览器进程。
8. 浏览器进程根据DrawQuad消息**生成页面**，并**显示**到显示器上。

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：

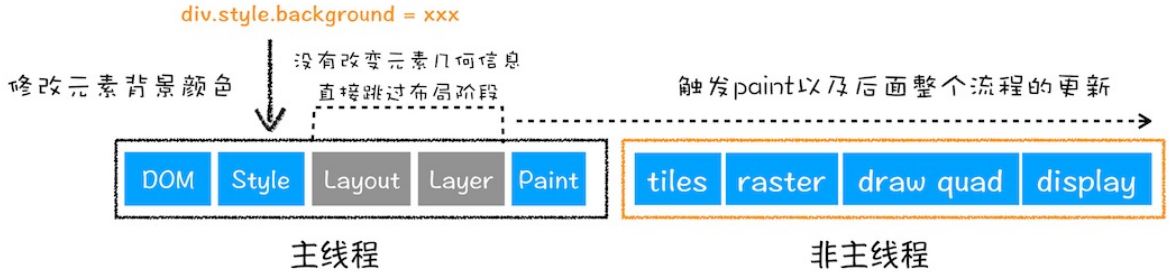


更新元素的几何属性

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：

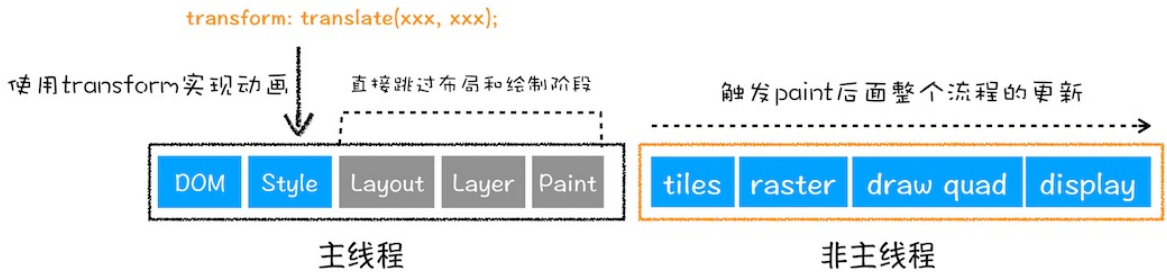


更新元素背景

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，**重绘省去了布局和分层阶段，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：



避开重排和重绘

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。

精选留言：

- mfist 2019-08-17 06:49:05
  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。
  [5赞]

  作者回复2019-08-17 10:22:18
  很赞

- ytd 2019-08-17 10:52:40
  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？ [3赞]

  作者回复2019-08-19 08:32:48
  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

- 杨陆伟 2019-08-17 15:52:49
  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 [2赞]

  作者回复2019-08-18 12:10:37
  说的好，如果感觉到复杂和无序，那一定是哪里错了

- 淡 2019-08-17 14:47:48
  老师，你好。
  关于把图层分块这块有些疑问，比如手机或者屏幕设备的尺寸并不总是256 * 256 或者 512 * 512 的整数倍。这样就会导致绘制的高度并不总是和内容刚好相等，表现为手机端有时候执行js拿到的html内容高度不

对，不知道您遇到过没有？如果有，有没有一些好的实践获取内容高度呢，谢谢。 [2赞]

- Hurry 2019-08-18 10:20:24
  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 [1赞]

  作者回复2019-08-19 08:20:01
  总结的很好，很有经验👍

- 郧 2019-08-17 09:04:05
  渲染进程里的帧的概念是什么样子的呢？一个page是一帧吗 [1赞]

  作者回复2019-08-19 08:38:59
  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。

- tokey 2019-08-17 00:32:50
  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% [1赞]

  作者回复2019-08-17 10:27:47
  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约简单高效！

- 李懂 2019-08-19 18:02:39
  我改变了宽高，是不是只对同一图层有影响，其他图层不会重排？

- 羽蝶曲 2019-08-19 16:51:02
  老师，我有个问题想问下，我试了https://developer.mozilla.org/zh-CN/docs/Web/Guide/CSS/Underst

anding_z_index/The_stacking_context 这里面z-index的例子，为啥Layers里看到的图层还是只有一个，不是说拥有层叠上下文属性的元素会被提升为单独的一层吗？
麻烦帮忙解惑，谢谢！

- Chao 2019-08-19 15:53:00
  获取下高宽都将触发重排， 所以本身很难避免。

- 😊 2019-08-19 08:29:45
  减少重排重绘可以减少gpu等的调用频次，gpu调用本身开销不小

  作者回复2019-08-19 09:33:37
  重绘，重排和GPU可没什么关系哦，你看文中介绍的，他们是cpu运算的，且占用页面主线程！

  所以如果没有这两个阶段的操作，那么每生成效率会高很多。

- 帅气小熊猫 2019-08-19 07:31:24
  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2019-08-19 09:40:03
  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。

- 安追 2019-08-18 21:30:35
  请教老师，因为重绘(paint)的输入是图层树，“重绘省去了布局和分层阶段”是否可以理解为：因为修改绘制属性并不改变元素的布局计算，所以重绘渲染的布局(layout)、分层(layer)两个阶段处理得非常快以至于可以忽略不计。

  作者回复2019-08-19 08:19:36
  不是处理非常快，而是直接跳过这些阶段，没处理。

- 安追 2019-08-18 20:40:22
  请问老师，切换标签页也是发出某个命令给浏览器进程的viz组件，然后直接取出内存中的已经绘制的图像来显示吗？

- tokey 2019-08-17 17:16:50
  老师的课程可以加个浏览器中的事件循环和js的事件机智么？

  作者回复2019-08-17 19:18:36
  循环系统是页面核心，当然要讲，而且还要大讲特讲。
  详细内容我放到了第三个模块中，你可以看下本专栏的目录结构。

- tokey 2019-08-17 12:24:29
  那您说的“通常”，可以举个例子么老师。

  还有您的课程会讲浏览器渲染帧么

- 许童童 2019-08-17 11:12:06
  为什么减少重绘、重排能优化 Web 性能吗？

宏观上来看，减少重绘、重排因为跳过了一些渲染流水线的步骤，使总体工作量减少了，所以性能得到了提升。

那又有那些具体的实践方法能减少重绘、重排呢?
尽量少用JS直接操作DOM改变宽高等信息，使用transform类的属性在单独的合成层上操作，有多个操作先在内存中合并，最后再一次性提交。

- ytd 2019-08-17 08:47:07
  重排和重绘都是渲染进程的主线程中进行的，减少这类操作可以减少主线程的资源占用，提高主线程绘制效率。在编写js时尽量减少dom操作或合并dom操作，dom操作需要重新生成dom树，如果影响布局就需要重新生成布局树，再重新生成分层树，再进行绘制。ps：感觉生成个页面好复杂呀，另外，以前从没注意过chrome开发者工具还有个Layers标签，chrome开发者工具真是一堆宝呀。

  作者回复2019-08-19 08:40:50
  开发者工具我们后续还要做详细分析