链滴

CSS 动画性能实践与优化

作者：Rabbitzzc

原文链接：https://ld246.com/article/1589518580970

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2020/05/image-75e6c513.png?imageView2/2/interlace/1/format/jpg"></>
<blockquote>
上次记得发个帖，说会记录一篇关于 CSS 动画性能的 Blog，工作太忙了，一直没时间去记录和践，总算是完成了！能力有限，当做为社区慢慢做点小力吧<img alt="doge" class="emoji" src="https://unpkg.com/vditor/dist/images/emoji/doge.png" title="doge">
</blockquote>
<hr>
随着 JavaScript 以及浏览器的发展，github 开源了很多 动画库进行动画处理，并且每个人似都会实现这些 动画库用作现成的动画解决方案。比如 Bounce.js、Anime.js 等等。 CSS 动画及其性。
适当的了解一下 CSS 动画也是必要的。CSS 基础动画依赖一些 CSS 属性，这些属性通常会在基于 CSS 的动画实现中频繁使用。常用的属性如下：

opacity
position(absolute /relative)
transform(translate)
left, right, top, bottom

在比较不同 CSS 属性动画性能之前，首先需要了解浏览器的渲染原理。
<h3 id="渲染树构建-布局及绘制">渲染树构建、布局及绘制</h3>
<h4 id="渲染进程">渲染进程</h4>
渲染页面就是让用户看见页面，核心目的在于，转换 HTML、CSS、JS 为用户可交互的 web 页。渲染进程中主要包含以下线程：

Main thread 主线程（负责解析，排版合成，Render Tree 绘制，JavaScript 执行等任务）
Compositor thread 合成线程（或者叫排版线程，负责将网页内部位图缓存/纹理输出到窗口的缓存，从而把网页显示在屏幕上。在使用 GPU 合成的情况下，也有可能只是产生 GL 绘图指令，然将绘图指令的缓存发送给 GPU 线程执行）
Worker thread 工作线程
Raster thread 光栅线程（如果主线程只负责将网页内容转换为绘图指令列表，光栅线程会栅格每一个图块而且把它们存储在 GPU 的内存中，光栅化的本质是坐标变换、几何离散化，然后再填充
GPU thread GPU 线程（如果使用 GPU 合成，则由 GPU 线程负责执行 GL 绘图指令）

<h3 id="CSS-渲染优化原理">CSS 渲染优化原理</h3>
浏览器对页面内容的渲染首先是渲染树的构建：

DOM 树与 CSSOM 树合并后形成渲染树（Render Tree）
渲染树只包含渲染网页所需的节点

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src=""><img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/fil/2020/05/image-76e72434.png?imageView2/2/interlace/1/format/jpg">
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2020/05/image-e3118404.png?imageView2/2/interlace/1/format/jpg"></>
如图，渲染树构建以后，浏览器需要做的动作为：Layout => Paint => Composite，具含义如下：

JavaScript JavaScript 实现动画效果，DOM 元素操作等
Recalculate Style（或 Style） 计算将要应用到元素上的样式
Layout 创建元素的布局并将其放置在屏幕上

Paint 在所有创建的布局中添加像素，更多的是为每个图层创建位图，GPU 使用此位图在屏幕上染图层
Composite Layers 最后，浏览器在屏幕上创建图层，构造一个图层堆栈。该堆栈的顶视图将看来像一个完整的网页，其中每个元素都有其自己的位置

Layout 与 Paint 都是耗性能的的过程，特别是 Layout，浏览器需要重新计算样式元素的样式和置（Recalculate Style ）。因此一般对于 CSS 的动画优化，都是尽量减少 Layout 和 Paint 操作，如尽量避免 table 布局，尽量统一修改样式，而不是串行修改等等，关于 CSS 哪些属性会引起 Layout 与 Paint，可以参考 https://csstriggers.com/。
<h3 id="CSS-transform-vs-left-top">CSS transform vs left-top</h3>
这里主要比较的是 transform 与 left-top，类似网站也有 https://lz5z.com/css3_hardware_speedup/。
首先编写基础的 HTML：
<pre><code class="highlight-chroma"><body>
  <div class="container">
    <h1>Top/Left Animation</h1>
    <div class="animate animate1"></div>
  </div>
  <div class="container">
    <h1>Transform Animation</h1>
    <div class="animate animate2"></div>
  </div>
</body>
</code></pre>
<h4 id="使用-transform">使用 transform</h4>
<pre><code class="highlight-chroma">
.animate {
  position: absolut;
  top: 100px;
  left: 30px;
  width: 100px;
  height: 100px;
  border-radius: 5%;
  background: #ff200;
}
.animate2 {
  animation: move 3s ease infinite;
}
@keyframes move1 {
  50% {

  transform: translate(100px, 100px);
  }
}
</code></pre>
<h4 id="使用-left-top">使用 left、top</h4>
<pre><code class="highlight-chroma">.animate1 {
  animation: move 3s ease infinite;
}
@keyframes move {
  50% {
    top: 200px;
    left: 130px;
  }
}
</code></pre>
<h4 id="性能检测">性能检测</h4>
通过观察上面两种不同的 CSS 动画实现方案，虽然实现了相同的功能，但当我们在 chrome dev-tool 中测量性能指标时，我们会看到两种情况在性能方面的实际差异。

通过上面两张图，两个球虽然看起来都在同一层上，但是，当我们以 3D 旋转方式移动该层时，们发现 transform 动画为另一个球创建了另一个图层。
对于 left-top，因为 Layout 与 Paint 发生在动画的每一帧；而对于 transform，两个图层发送给 GPU，GPU 中 transform 是不会触发 repaint 的，因此动画执行的过程只是两个图层之间的相对移，这也是 GPU 硬件加速的优势，能够非常快地合成完整的网页。

通过突出显示主线程的单个任务，在两种情况下，任务执行的活动是不同的。top-left 会有 Layout , Painting 和 Recalculate style，而 transform 则是由 GPU 执行的单个任务，无需在 DOM 中呈任何内容即可移动 Composite Layers。

通过上面左图（left-top），解释了我们的主线程忙于布局和样式重新计算的确切程度。因此，以借助 GPU 的强大功能，我们可以使主线程保持空闲状态，这有助于我们提高 web 应用的性能。</
>
<h4 id="分析">分析</h4>
通过上面的实践，可以看到 CSS transform 属性实际上是创建了独立的图层，利用 GPU 硬件加，仅仅发生了 Composite ，实现的平移动画更加平滑，性能也更好。类似的 CSS 属性还有 opacity

<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https:/b3logfile.com/file/2020/05/image-e6f7886a.png?imageView2/2/interlace/1/format/jpg"></>
<h3 id="选择-GPU-加速">选择 GPU 加速</h3>
上述实践可以看到，CSS transform 属性实际上是创建了独立的图层，触发了硬件加速。此方法优点是，定期重绘的或通过变形在屏幕上移动的元素，可以在不影响其他元素的情况下进行处理。Sketch、GIMP 或 Photoshop 之类的文件也是如此，各个层可以在彼此的上面处理并合成，以创建最终像页面。其他类似的属性还有 opacity、filter、will-change，创建新层的最佳方式是使用 will-chang 属性。
<pre><code class="highlight-chroma">.demo {
  will-change: tranform;
}
</code></pre>
对于不支持 will-change 但受益于层创建的浏览器，例如 Safari 和 Mobile Safari，需要使用 3D 变形来强制创建一个新层：
<pre><code class="highlight-chroma">.demo {
  transform: translteZ(0);
}
</code></pre>
但需要注意的是不要创建太多层，因为每层都需要内存和管理开销（像素需要存储），GPU 也要缓存它们以便于后续动画的使用。具体原则如下：

坚持使用 transform 和 opacity 属性更改来实现动画
使用 will-change 或 translateZ 提升移动的元素
避免过度使用提升规则（各层都需要内存和管理开销）

<blockquote>
即如无必要，请勿提升元素创建新的图层
</blockquote>
<h3 id="总结">总结</h3>
从渲染过程以及复合图层（Composite Layers），可以总结一下几点来提升 CSS 动画性能：</>

尽量避免 reflow 与 repaint
CSS 动画尽量使用 transform 与 opacity，比如 animate.css 以及 anime.js 首页提示
保持主线程做最正确的事，尽量避免使用 JS 动画

参考链接
<hr>

https://csstriggers.com/
https://developers.google.com/web/fndamentals/performance/critical-rendering-path/render-tree-construction?hl=zh-cn</li

<a href="https://ld246.com/forward?goto=https%3A%2F%2Fdevelopers.google.com%2web%2Ffundamentals%2Fperformance%2Frendering%2Fsimplify-paint-complexity-and-redu

e-paint-areas%3Fhl%3Dzh-cn" target="_blank" rel="nofollow ugc">https://developers.google
com/web/fundamentals/performance/rendering/simplify-paint-complexity-and-reduce-paint
areas?hl=zh-cn
<a href="https://ld246.com/forward?goto=https%3A%2F%2Fwww.smashingmagazine.c
m%2F2016%2F12%2Fgpu-animation-doing-it-right%2F" target="_blank" rel="nofollow ugc"
https://www.smashingmagazine.com/2016/12/gpu-animation-doing-it-right/
<a href="https://ld246.com/forward?goto=https%3A%2F%2Fwww.html5rocks.com%2Fe
%2Ftutorials%2Fspeed%2Fhigh-performance-animations%2F" target="_blank" rel="nofollow
gc">https://www.html5rocks.com/en/tutorials/speed/high-performance-animations/</l
>

原文链接：CSS 动画性能实践与优化