高性能动画

为什么有些动画慢，怎样让它快。

heyw

2022/12/20

heyw

随时可能出现的地球生命体，闪烁技能 ✨ 点满

白天是个前端

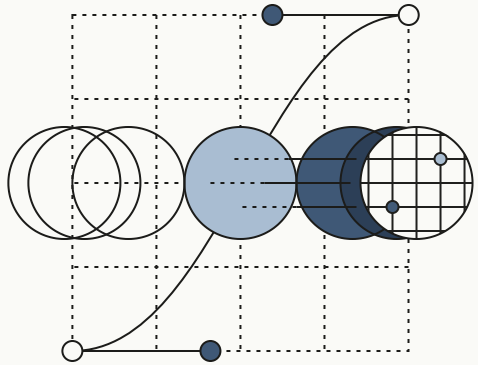
积极参与开源社区

曾在 @Ctrip, Inc. 实习

heyw.me

eggx8

1. 浏览器怎样渲染动画
2. 渲染管线
3. 合成属性
4. CSS 还是 JavaScript
5. 写出高性能的 CSS 动画
6. 避开触发布局与绘制的属性
7. 用 will-change 强制建层
8. 两个例子
9. 检查动画是否触发布局
10. 检查动画是否丢帧
11. 检查动画是否触发绘制
12. 检查动画是否创建图层
13. 一次真实的优化



浏览器怎样渲染动画

现代浏览器里，只有两个属性可以低成本地做动画：`transform` 和 `opacity`。

流畅的标准是 60 FPS——每一帧只有约 16.7ms 的预算， 一帧里的活干不完，人眼就能看出卡顿。

渲染管线

把内容显示到屏幕上，浏览器要依次走完四步

1. 样式（Style）：计算每个元素应用到的样式
2. 布局（Layout）：算出每个元素的几何尺寸和位置
3. 绘制（Paint）：把元素的像素画进图层
4. 合成（Composite）：把各个图层拼到屏幕上

这四步是顺序的：动画改到哪一步，后面的步骤就得全部重做。 所以改布局的动画最贵，只改合成的动画最便宜。

延伸阅读

- From Braces to Pixels
- Inside look at a modern web browser

合成属性

合成（compositing）是把页面拆成若干图层、各自光栅化成像素，再拼装出最终画面的过程。

`opacity` 之所以便宜，是因为只要元素在自己的图层里， 对它的改动就可以完全交给 GPU 在合成这一步处理，前面三步一步都不用走。

基于 Chromium 和 WebKit 的浏览器，会给任何带 `transform` 或 `opacity` 动画的元素单独建一个图层。

延伸阅读

- GPU Animation: Doing It Right

CSS 还是 JavaScript?

从性能角度看，用哪个做动画更好?

CSS 动画和 Web Animations API（在支持的浏览器里） 通常跑在**合成器线程**上，与执行样式、布局、绘制和 JavaScript 的主线程互不干扰。

主线程被耗时任务堵住时，这些动画照常播放。

很多情况下，`transform` 和 `opacity` 的其它变化也能由合成器线程接管。

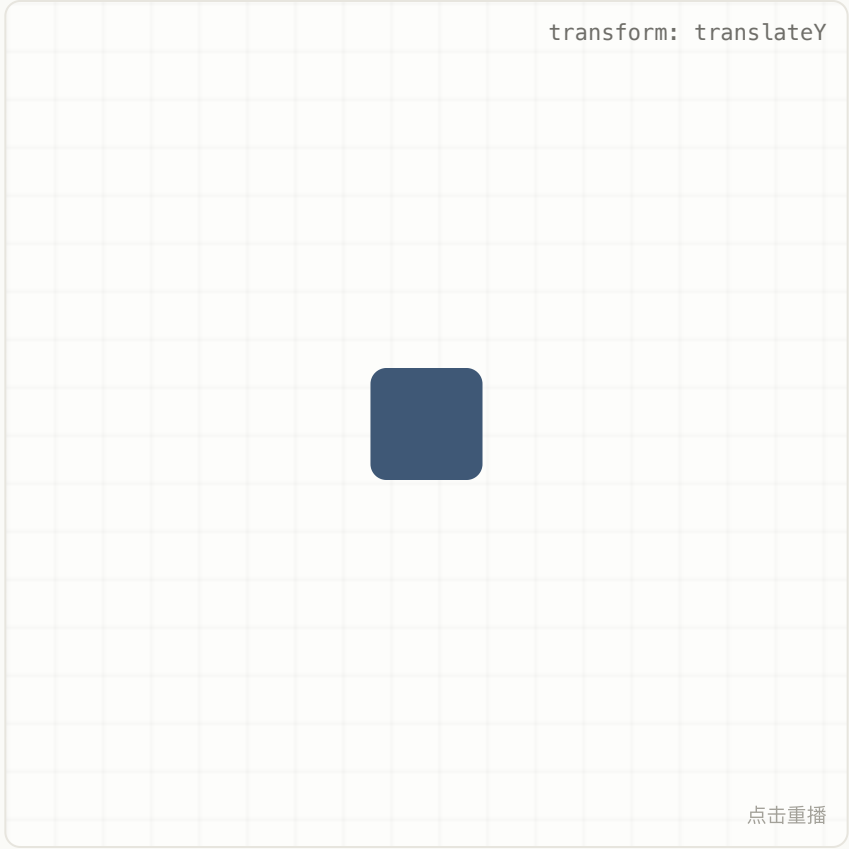
写出高性能的 CSS 动画

`transform` `opacity` `will-change`

移动元素

移动元素，用 `transform` 的 `translate` 或 `rotate`， 不要动 `top` / `left` / `margin`。

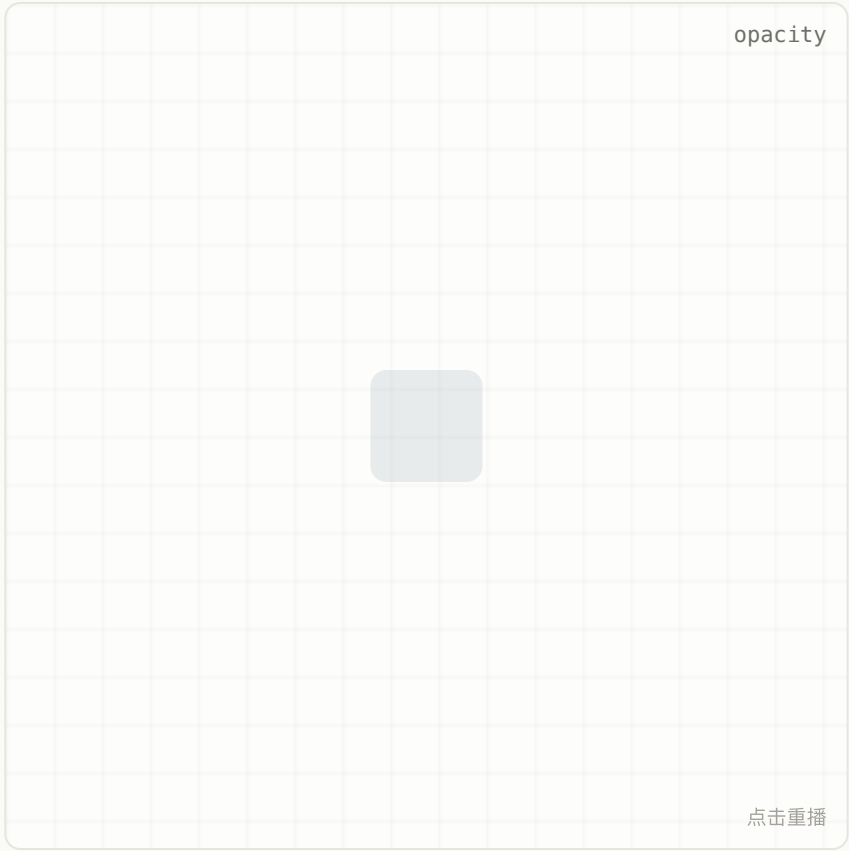
```
.animate {
  animation: slide-in 0.7s both;
}

@keyframes slide-in {
  0% {
    transform: translateY(-1000px);
  }
  100% {
    transform: translateY(0);
  }
}
```



显示与隐藏

让元素显形或消失，用 `opacity`。

```
.animate {
  animation: fade 2.5s both;
}

@keyframes fade {
  0% {
    opacity: 1;
  }
  50% {
    opacity: 0;
  }
  100% {
    opacity: 1;
  }
}
```



● Layout ● Paint ● Composite

	Change from default				Subsequent updates			
	Blink	Gecko	WebKit	EdgeHTML	Blink	Gecko	WebKit	EdgeHTML
align-content								
align-items								
align-self								
backface-visibility								
background-attachment								
background-blend-mode								
background-clip								
background-color								
background-image								
background-origin								
background-position								
background-repeat								
background-size								

避开触发布局与绘制的属性

把 `transform` 和 `opacity` 之外的属性用于动画之前， 先查清楚它会触发渲染管线的哪几步。

除非别无选择，否则不要用会触发布局或绘制的属性做动画。

上一页的 CSS Triggers 就是干这个的： 每个属性改动后走到管线哪一步，查一下就知道。

用 will-change 强制建层

把元素放进新图层后，重绘它就不用再牵动页面的其它部分。

浏览器通常自己就能判断哪些元素该单独建层，判断失误的时候， 可以用 `will-change` 手动声明：这个元素接下来会变。

transform: translateZ(0)

⚠ 注意

建层本身有内存开销，别把它当成先手优化。

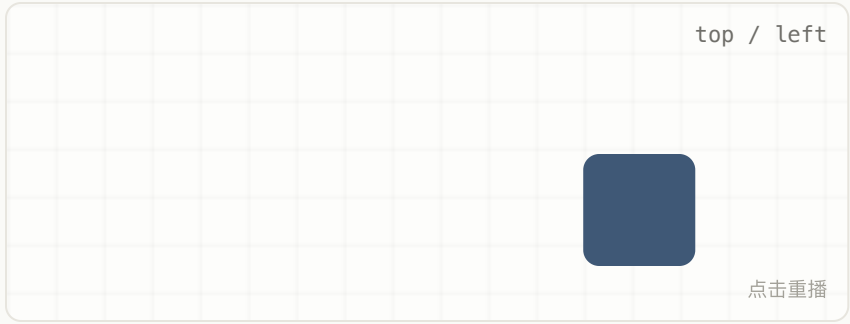
只有确实看到了卡顿、并且判断建层能救，才用它。

延伸阅读

- Everything You Need To Know About The CSS will-change Property

同一段位移，两种写法

```css
.box {
  position: absolute;
  top: 12px;
  left: 12px;
  animation: move 3s ease infinite;
}

@keyframes move {
  50% { top: calc(100% - 68px);
        left: calc(100% - 68px); }
}
```



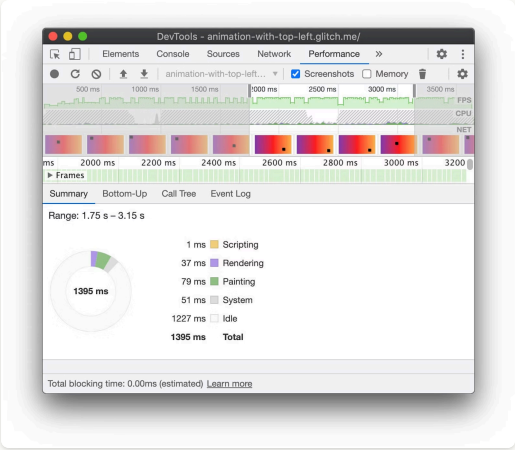

```css
.box {
  animation: move 3s ease infinite;
}


@keyframes move {
  50% { transform: translate(calc(100cqw - 80px),
                              calc(100cqh - 80px)); }
}
```

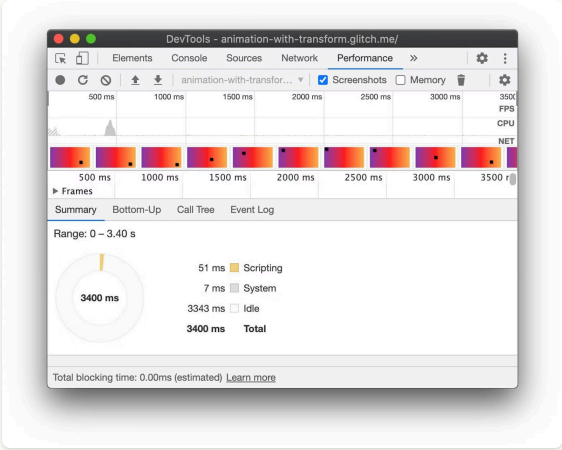
肉眼看不出差别——差别要用工具量出来，这正是下一节要做的事。

找出慢动画：检查是否触发布局

1. 打开 DevTools 的 **Performance** 面板
2. 在动画播放时录制一段运行时性能
3. 查看 **Summary** 选项卡

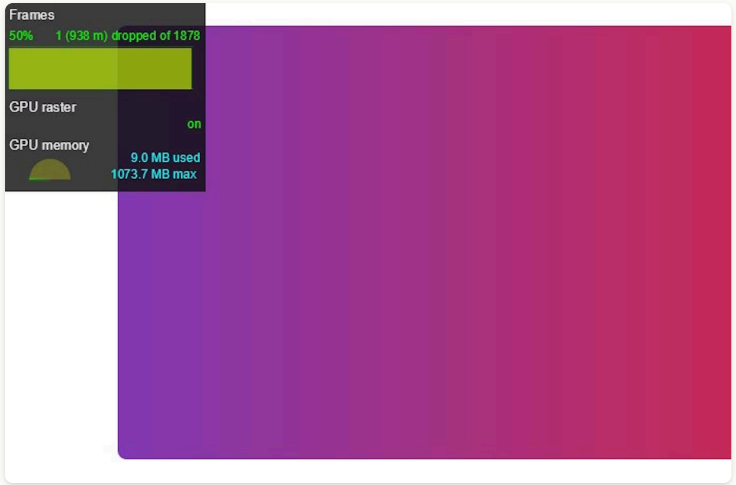
Summary 里 Rendering 的数值不是零，就说明这个动画在让浏览器做布局。

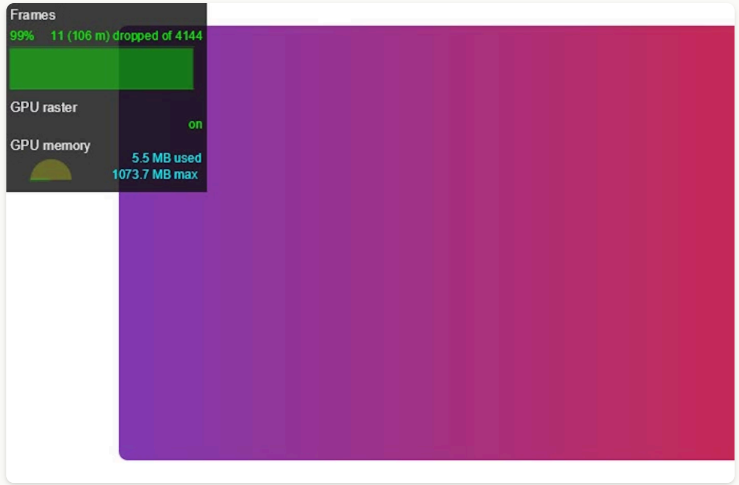




找出慢动画：检查是否丢帧

1. 打开 DevTools 的 **Rendering** 选项卡
2. 勾选 **FPS meter**
3. 在动画播放时观察数值

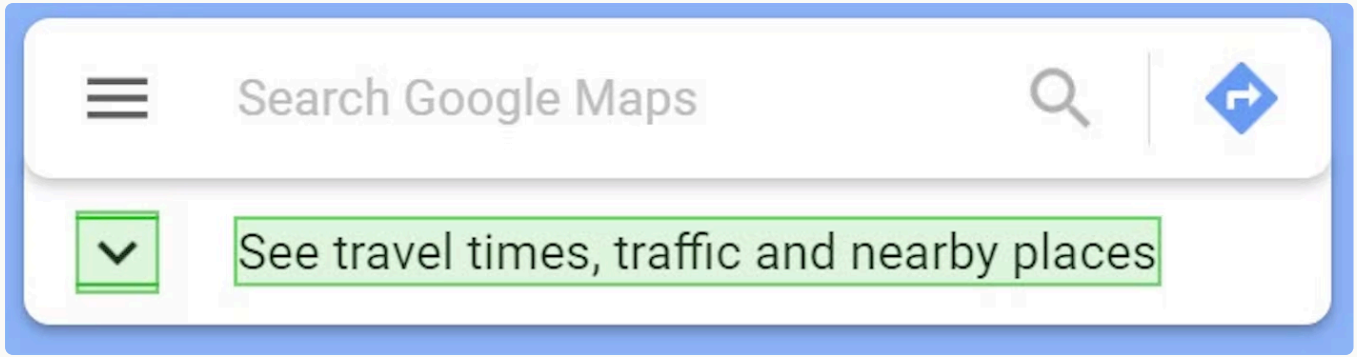




找出慢动画：检查是否触发绘制

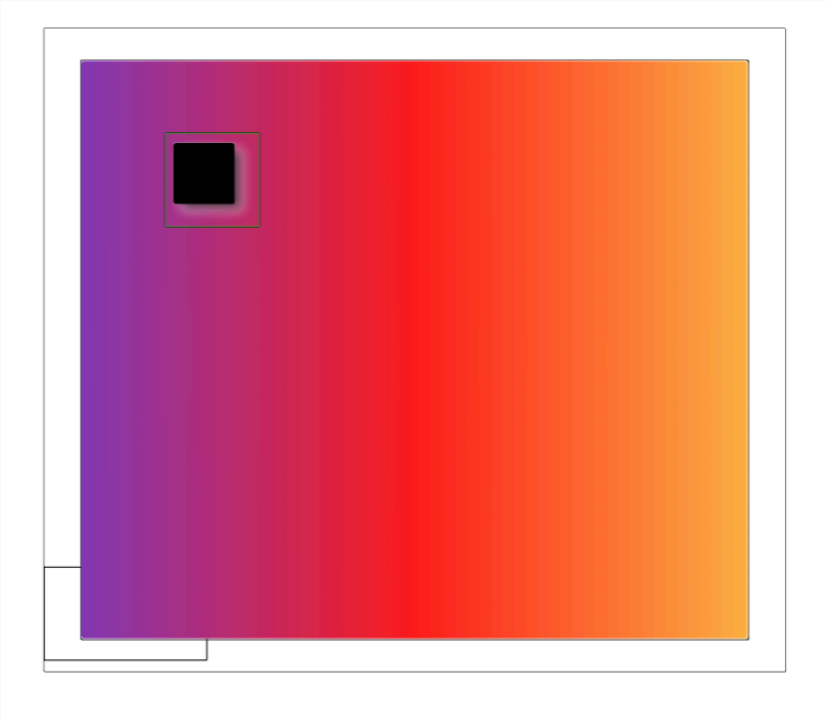
绘制里也分贵贱：涉及模糊的东西（比如阴影）比画一个纯色方块贵得多。`background: red` 和 `box-shadow: 0 4px 4px rgba(0,0,0,.5)` 在 CSS 里看不出差别，在绘制耗时上差别很大。

1. 打开 DevTools 的 **Rendering** 选项卡
2. 勾选 **Paint flashing**
3. 触发动画，看哪些区域在闪绿



找出慢动画：检查图层

1. 打开 DevTools 的 **Layers** 选项卡
2. 在左侧列表里逐个点开图层，看它为什么被创建



一次真实的优化

一个业务需求里的四处动画改动，改法全部来自前面的原则。

改动一 · 二：位移与宽度

margin-left 改成 translateX；width 生长改成 scaleX，配 transform-origin: left

```
@keyframes right-to-left {
  from {
-    margin-left: -22px;
+    transform: translateX(22px);
    opacity: 0;
  }
  to {
-    margin-left: -54px;
+    transform: translateX(0);
    opacity: 1;
  }
}
```

```
.bottom-line {
+ width: 97px;
+ transform-origin: left;
  animation: bottom-line-grow 0.6s forwards;
}

@keyframes bottom-line-grow {
  from {
-   width: 0;
+   transform: scaleX(0);
  }
  to {
-   width: 97px;
+   transform: scaleX(1);
  }
}
```

改动三 · 四：阴影与建层

与其动 box-shadow，不如动一个带阴影的透明伪元素的 opacity

```
.card {
+ position: relative;
- transition: box-shadow 0.15s ease-out;

- &:hover {
-   box-shadow: $shadow-hover;
+ &::after {
+   position: absolute;
+   inset: 0;
+   box-shadow: $shadow-hover;
+   content: '';
+   opacity: 0;
+   transition: opacity 0.15s ease-out;
+ }
+ &:hover::after {
+   opacity: 1;
  }
}
```

```
+ transform: translateZ(0);
```

最后在个别元素上强制建了层。 前提是建层的内存代价，小于动画本身的代价。

想继续深入，请查阅 web.dev 与 developer.chrome.com

本次分享的原理部分即整理自这两处的文档。

谢谢

幻灯片可在 heyw.me/talks 查看与下载