css图层

浏览器在渲染一个页面时，会将页面分为很多个图层，图层有大有小，每个图层上有一个或多个节点。

在渲染DOM的时候，浏览器所做的工作实际上是：

1. 获取DOM后分割为多个图层
2. 对每个图层的节点计算样式结果 （Recalculate style--样式重计算）
3. 为每个节点生成图形和位置 （Layout--布局，重排,回流）
4. 将每个节点绘制填充到图层位图中 （Paint--重绘）
5. 图层作为纹理上传至GPU
6. 组合多个图层到页面上生成最终屏幕图像 （Composite Layers--图层重组）

图层创建的条件

Chrome浏览器满足以下任意情况就会创建图层：

1. 拥有具有3D变换的CSS属性
2. 使用加速视频解码的

节点

3.
4. CSS3动画的节点
5. 拥有CSS加速属性的元素(will-change)

重绘(Repaint)

重绘是一个元素外观的改变所触发的浏览器行为，例如改变outline、背景色等属性。浏览器会根据元素的新属性重新绘制，

使元素呈现新的外观。重绘不会带来重新布局，所以并不一定伴随重排。

需要注意的是：重绘重排都是以图层为单位，如果图层中某个元素需要重绘，那么整个图层都需要重绘。

所以为了提高性能，我们应该让这些“变化的东西”拥有一个自己一个图层，

不过好在绝大多数的浏览器自己会为CSS3动画的节点自动创建图层。

重排(Reflow 又称：回流)

渲染对象在创建完成并添加到渲染树时，并不包含位置和大小信息。计算这些值的过程称为布局或重排(回流)。

> "重绘"不一定需要"重排"，比如改变某个网页元素的颜色，就只会触发"重绘"，不会触发"重排"，因为布局没有改变。
> "重排"大多数情况下会导致"重绘"，比如改变一个网页元素的位置，就会同时触发"重排"和"重绘"，因为布局改变了。

触发重绘的属性

- color
- border-style
- border-radius
- visibility
- text-decoration
- background
- background-image
- background-position
- background-repeat
- background-size
- outline-color
- outline
- outline-style
- outline-width
- box-shadow

触发重排(回流)的属性

- width
- height
- padding
- margin
- display
- border-width
- border
- min-height
- top
- bottom
- left
- right
- position
- float
- clear
- text-align
- overflow-y
- font-weight
- overflow
- font-family
- line-height
- vertival-align
- white-space

常见的触发重排的操作

Reflow(重排) 的成本比 Repaint(重绘) 的成本高很多很多。

一个结点的 Reflow 很有可能导致子结点，甚至父点以及同级结点的 Reflow。

在一些高性能的电脑上也许还没什么，但是如果 Reflow 发生在手机上，那么这个过程是非常痛苦和耗电的。

> 所以，下面这些动作有很大可能会是成本比较高的。
> 当你增加、删除、修改 `DOM` 结点时，会导致 `Reflow , Repaint`。
> 当你移动 `DOM` 的位置
> 当你修改 `CSS` 样式的时候。
> 当你 `Resize` 窗口的时候（移动端没有这个问题，因为移动端的缩放没有影响布局视口)
> 当你修改网页的默认字体时。
> 【获取某些属性时(width,height...)！！！！！】
> 注：`display:none` 会触发 `reflow`，而 `visibility:hidden` 只会触发 `repaint`，因为没有发生位置变化。

优化方案（重绘重排）

我们已知：浏览器渲染页面时经历了如下“细致”的环节：

```
    1. 计算需要被加载到节点上的样式结果（Recalculate style--样式重计算）
    2. 为每个节点生成图形和位置（Layout--重排或回流）
    3. 将每个节点填充到图层中（Paint--重绘）
    4. 组合图层到页面上（Composite Layers--图层重组）
        如果我们需要提升性能，需要做的就是减少浏览器在运行时所需要做的工作，即：尽量减少1234步。
```

```
【具体优化方案如下】：
1.元素位置移动变换时尽量使用CSS3的transform来代替对top left等的操作
    变换（transform）和透明度（opacity）的改变仅仅影响图层的组合
2.【使用opacity来代替visibility】
    (1).使用visibility不触发重排，但是依然重绘。
    (2).直接使用opacity即触发重绘，又触发重排（GPU底层设计如此！）。
    (3).opacity配合图层使用，即不触发重绘也不触发重排。
        原因：
        透明度的改变时，GPU在绘画时只是简单的降低之前已经画好的纹理的alpha值来达到效果，并不需要整体的重绘。
        不过这个前提是这个被修改opacity本身必须是一个图层。
3.【不要使用table布局】
    table-cell
4.将【多次改变样式属性的操作合并成一次】操作
    不要一条一条地修改DOM的样式，预先定义好class，然后修改DOM的className
5.【将DOM离线后再修改】
    由于display属性为none的元素不在渲染树中，对隐藏的元素操作不会引发其他元素的重排。
    如果要对一个元素进行复杂的操作时，可以先隐藏它，操作完成后再显示。这样只在隐藏和显示时触发2次重排。
6.【利用文档碎片】(documentFragment)------vue使用了该种方式提升性能。
7.【不要把获取某些DOM节点的属性值放在一个循环里当成循环的变量】
    当你请求向浏览器请求一些 style信息的时候，就会让浏览器flush队列，比如：
        1. offsetTop, offsetLeft, offsetWidth, offsetHeight
        2. scrollTop/Left/Width/Height
        3. clientTop/Left/Width/Height
        4. width,height
    当你请求上面的一些属性的时候，浏览器为了给你最精确的值，需要刷新内部队列，
    因为队列中可能会有影响到这些值的操作。即使你获取元素的布局和样式信息跟最近发生或改变的布局信息无关，
    浏览器都会强行刷新渲染队列。
8.动画实现过程中，启用GPU硬件加速:transform: tranlateZ(0)
9.为动画元素新建图层,提高动画元素的z-index
10.编写动画时，尽量使用如下的API:
```

requestAnimationFrame----请求动画帧

1. window.requestAnimationFrame()

** 与定时器的不同：定时器需要指定时间，而requestAnimationFrame根据显示器的刷新率执行，不需要指定时间。

说明：该方法会告诉浏览器在下一次重绘之前调用你所指定的函数

1. 参数：该方法使用一个回调函数作为参数，这个回调函数会在浏览器下一次重绘之前调用。

回调函数会被自动传入一个参数，DOMHighResTimeStamp，标识requestAnimationFrame()开始触发回调函数的当前时间

2. 返回值：一个 long 整数，请求 ID ，是回调列表中唯一的标识。是个非零值，没别的意义。你可以传这个值给 window.cancelAnimationFrame() 以取消回调函数。

备注：若你想在浏览器下次重绘之前继续更新下一帧动画，那么回调函数自身必须再次调用window.requestAnimationFrame()

2. window.cancelAnimationFrame(requestID)

取消一个先前通过调用window.requestAnimationFrame()方法添加到计划中的动画帧请求。

requestID是先前调用window.requestAnimationFrame()方法时返回的值，它是一个时间标识，用法与定时器的id类似。