浏览器渲染原理解析

- 解析HTML（Html Parser）
- 构建DOM树（DOM Tree）
- 构建CSSOM树（Style）
- 构建渲染树（Render Tree）
- 页面布局（Layout）
- 绘制渲染树（Painting）

布局

布局也称为重排或回流，布局流程输出的是一个“盒模型”，它会精确地捕获每个元素在视口内的精确位置和尺寸，HTML就是采用基于流的布局模型，页面元素的变动往往可能导致回流的发生，而回流的频发发生亦是影响页面性能的重要因素，另外，处于流后置位通常不会影响前置位的几何特征，故对后置位的修改往往比对前置位的修改对页面整体的影响要低。

绘制

绘制即是对DOM所分割的层（layer）进行对应的绘制，页面的回流一般都会伴随着重绘，但重绘行为的出现不一定伴随回流。

渲染层

我们平面直观所见到的图像是基于空间图层的重叠得到的，一般来说，拥有相同坐标空间的节点属于同一个渲染层。渲染层最初是用来实现层叠上下文，以此来保证页面元素以正确的顺序合成（composite），实现半透明重叠等效果。

创造渲染层的条件：

1. 根元素（html）
2. 有明确的position属性（fixed，relative，sticky，absolute）
3. 透明的（opacity小于1）
4. 有css滤镜（filter）
5. 有css mask属性
6. 当前有对于opacity，transform，filter，background-filter应用动画
7. overflow不为visible

合成层

合成层是特殊的渲染层，每个合成层有单独的绘图层，绘图层中的绘图上下文负责输出该层的位图，位图储存在共享内存中，作为纹理上传到GPU，最后由GPU将多个位图进行合成，最后绘制到屏幕上，而相对于合成层，一般的渲染层是和其第一个拥有绘图层的父层共用一个的绘图层的，提升为合成层后当需要repaint或reflow本身，不影响其它层，另外，合成层的位图会直接交由GPU合成处理，效率比CPU高。

影响页面性能的操作及优化分析

频繁操作DOM元素

使用js脚本频繁地操作DOM元素是影响页面性能的一大因素，频繁地对DOM进行操作可能导致页面重绘和回流的频繁发生，从而导致页面卡顿和性能消耗问题，从细节上可按如下方法进行优化：

1. 使用文档片段

```
var fragment = document.createDocumentFrament();
// 一些基于fragment的大量DOM操作
......
document.getElement('element').appendChild(fragment);
```

2.设置DOM元素的display为none在操作元素

```
var myElement = document.getElementById('myElement');
        myElement.style.display = 'none';
    //一些基于myElement的大量DOM操作
    ......
    myElement.style.display = 'block';
```

3.复制DOM元素到内存中再对其进行操作

```
var old = document.getElementById('myElement');
var clone = old.cloneNode(true);
//一些基于clone的大量操作
......
old.parentNode.replaceChild(clone, old);
```

4.用局部变量缓存样式信息从而避免频繁的获取DOM数据

```
//bad operation
for (var i = 0; i < paragraphs.length; i++){
    paragraphs[i].style.width = box.offsetWidth + 'px';
}
//better operation
var width = box.offsetWidth;
for (var i = 0; i < paragraphs.length; i++){
    paragraphs[i].style.width = width + 'px';
}
```

5.合并多次DOM操作

```
//bad operation
var left = 10, top = 10;
```

```
el.style.top = top;
el.style.left = left;
//better operation
el.style.cssText += "; left: " + left + "px; top: " + top + "px;";
//better operation（将样式内容设置于某一类名，再进行元素类名绑定）
el.className += " theclassName";
```

6.css动画造成页面不流畅问题分析优化

使用css3动画造成页面的不流畅和卡顿问题，其潜在原因往往还是页面的回流和重绘，减少页面动画元素对其他元素的影响是提高性能的根本方向，而实现可如下：

1. 设置动画元素Position样式为absolute或fixed，可避免动画的进行对页面其他元素造成影响，导致重排和重绘的的发生；

2. 避免使用margin，top，right，left，width，height等属性执行动画，用transform进行代替；

3. 总而言之，尽量用transform和opacity完成动画的展示，因为这两个属性可以避免重排和重绘的发生。

4. 合理提升合成层，以减少页面不必要的绘制和重排。合成层的好处是不会影响到其他元素的绘制和不被其他层所影响，因此，为了彼此之前的影响造成的性能损失，我们需合理的将动画效果中的元素或固定元素提升为合成层。**提升合成层的最好方式是使用 CSS 的 will-change 属性。将will-change 设置为 opacity、transform、top、left、bottom、right 可以将元素提升为合成层。对于还不兼容该属性的浏览器，我们使用3D transform予以代替：**

```
#target {
  transform: translateZ(0);
}
```

合成层的提升也意味着性能的消耗增加，我们必须通过调试以测出合理的临界值，不能盲目提升合成层，此外，盲目提升合成层也可能造成重叠产生的额外合成层，容易导致层爆炸的出现，即页面连锁出现大量合成层默认提升，建议用google的timeline进行监控调试，避免出现不必要的意外消耗。