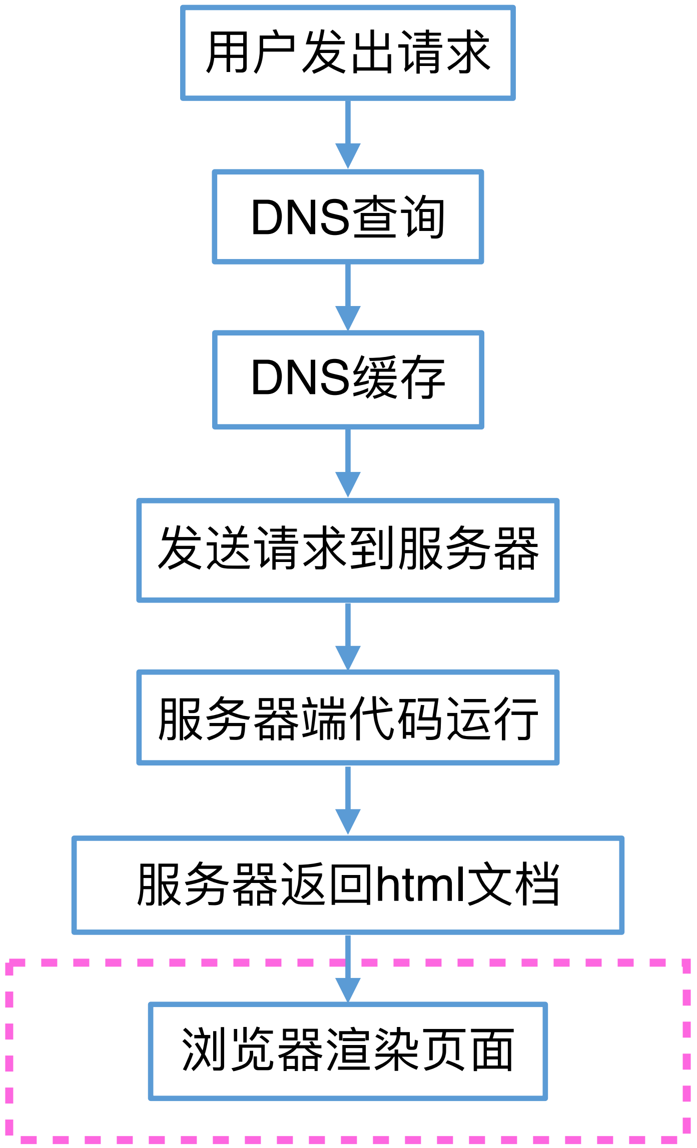
五阿哥
MINMETALS & ALIBABA

性能优化之浏览器渲染原理

从页面请求到页面渲染



用户发出请求
DNS查询
DNS缓存
发送请求到服务器
服务器端代码运行
服务器返回html文档
浏览器渲染页面

浏览器渲染引擎

IE (Trident)

Chrome (Blink)

Firefox (Gecko)

Opera (Blink)

Safari (Webkit)

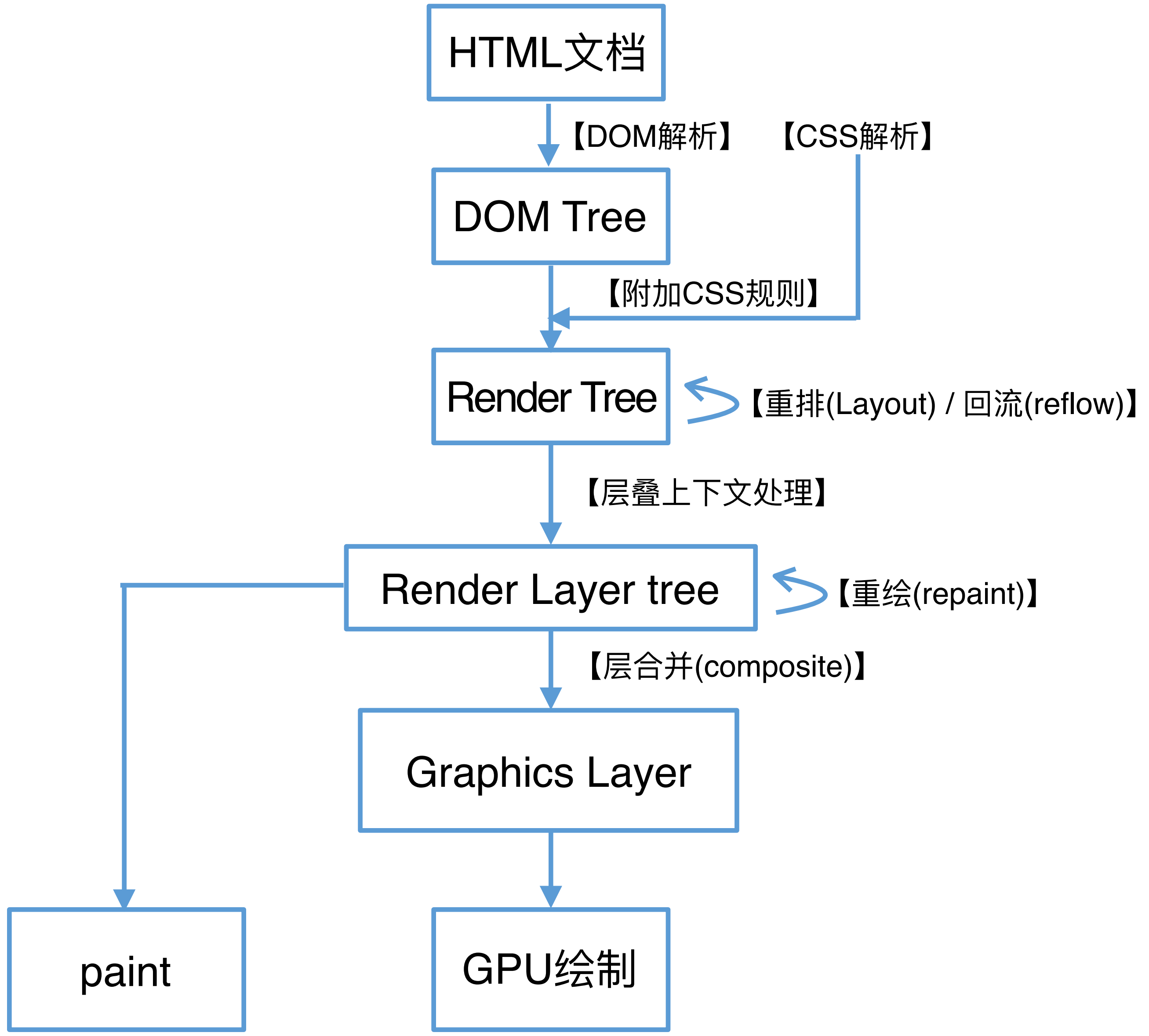
UC (U3)

QQ browser/ Wechat webview (X5/Blink)

五阿哥 MINMETALS & ALIBABA

页面渲染

页面渲染有两条路径：
一是软件路径
二是硬件路径（传说中的硬件加速）



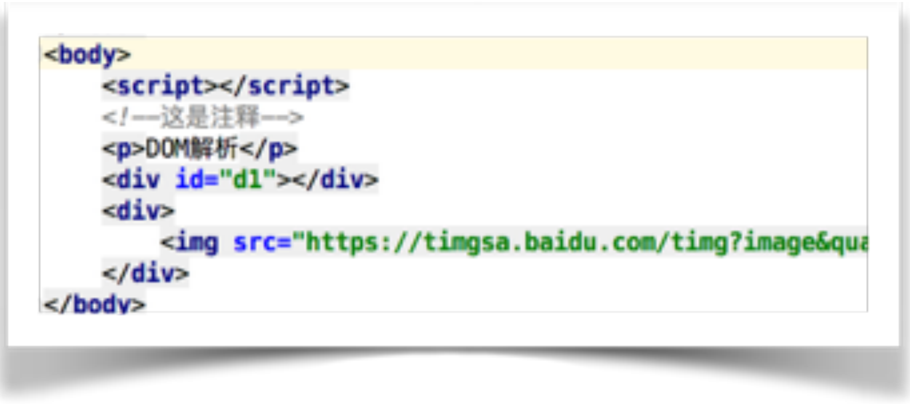
DOM解析

把HTML文档解析为DOM树的过程 Turn Html text to document object model

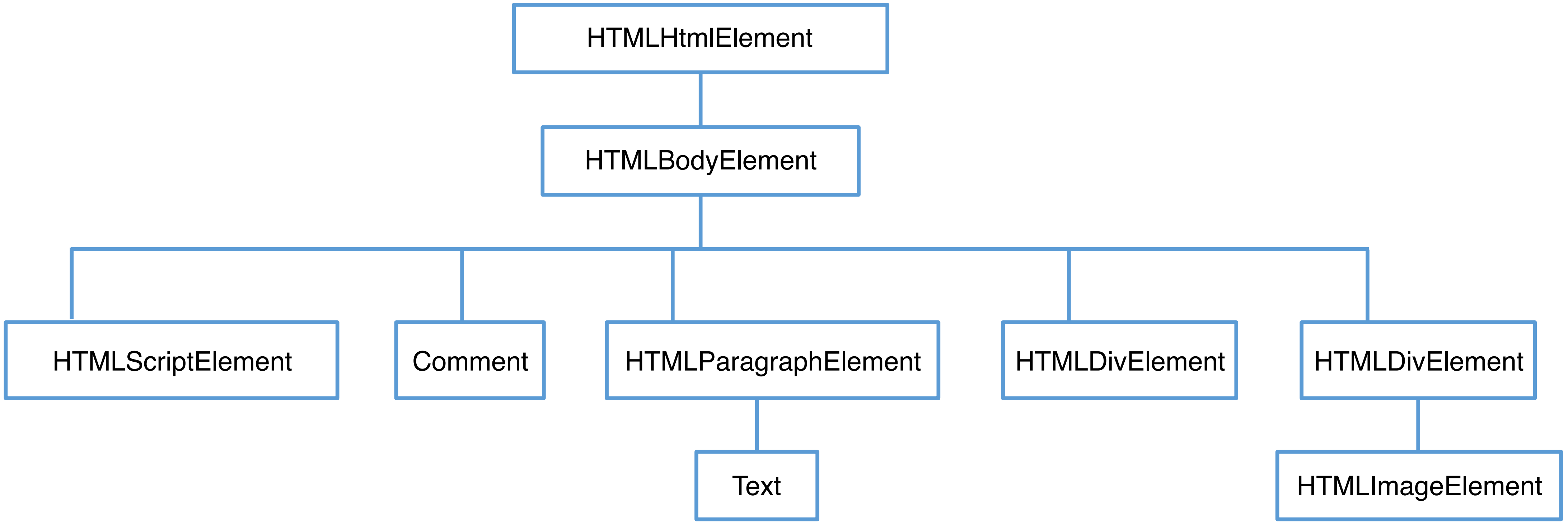
1、遇到<script>标签则停止解析，先执行js (unless explicitly declared as async)

2、此时图片资源并未加载完成

DOM Tree



```
<body>
    <script></script>
    <!--这是注释-->
    <p>DOM解析</p>
    <div id="d1"></div>
    <div>
        <img src="https://timgsa.baidu.com/timg?image&qua
    </div>
</body>
```



DOM Tree

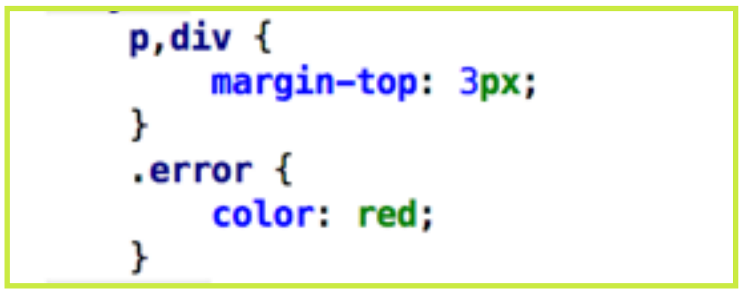
DOM树结构与HTML标签一一对应

1、display: none的元素也在DOM树中

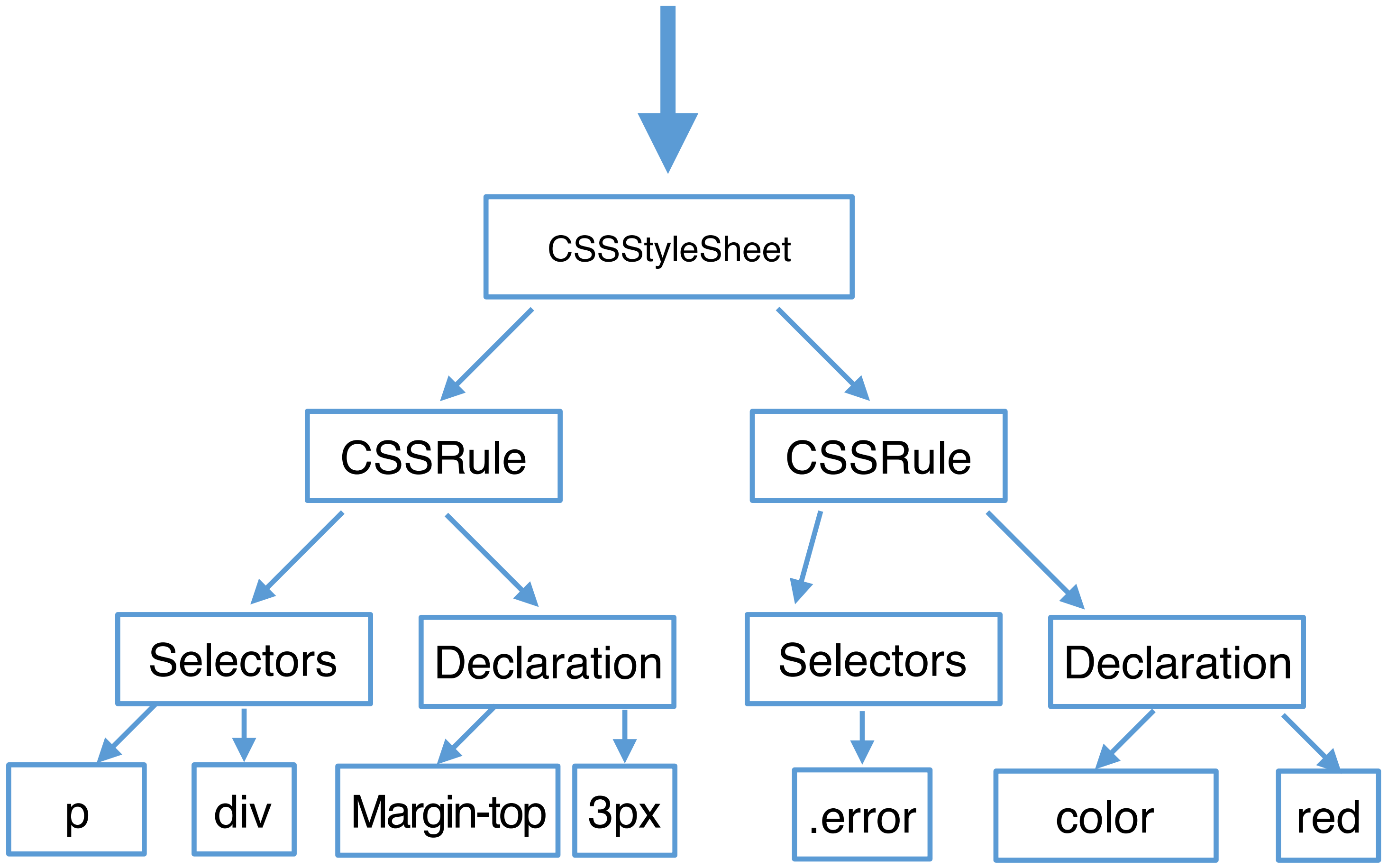
2、<script>标签在DOM树中

3、注释也在DOM树中

CSS解析

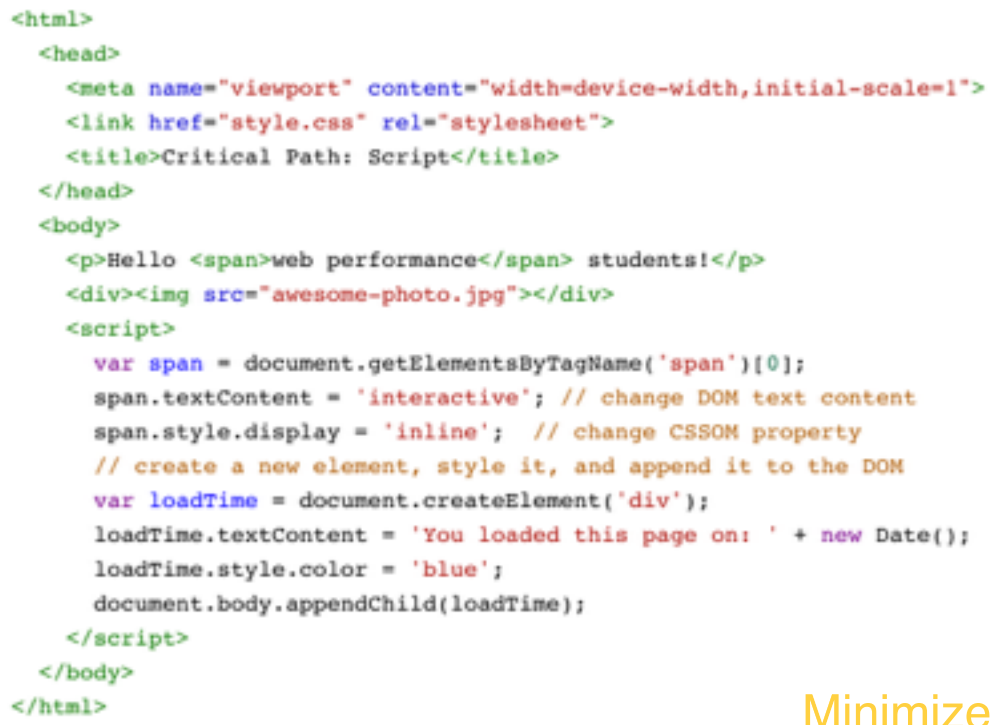


```
p,div {
    margin-top: 3px;
}
.error {
    color: red;
}
```



CSS解析

将CSS代码解析为CSS规则树的过程：与DOM解析同步进行



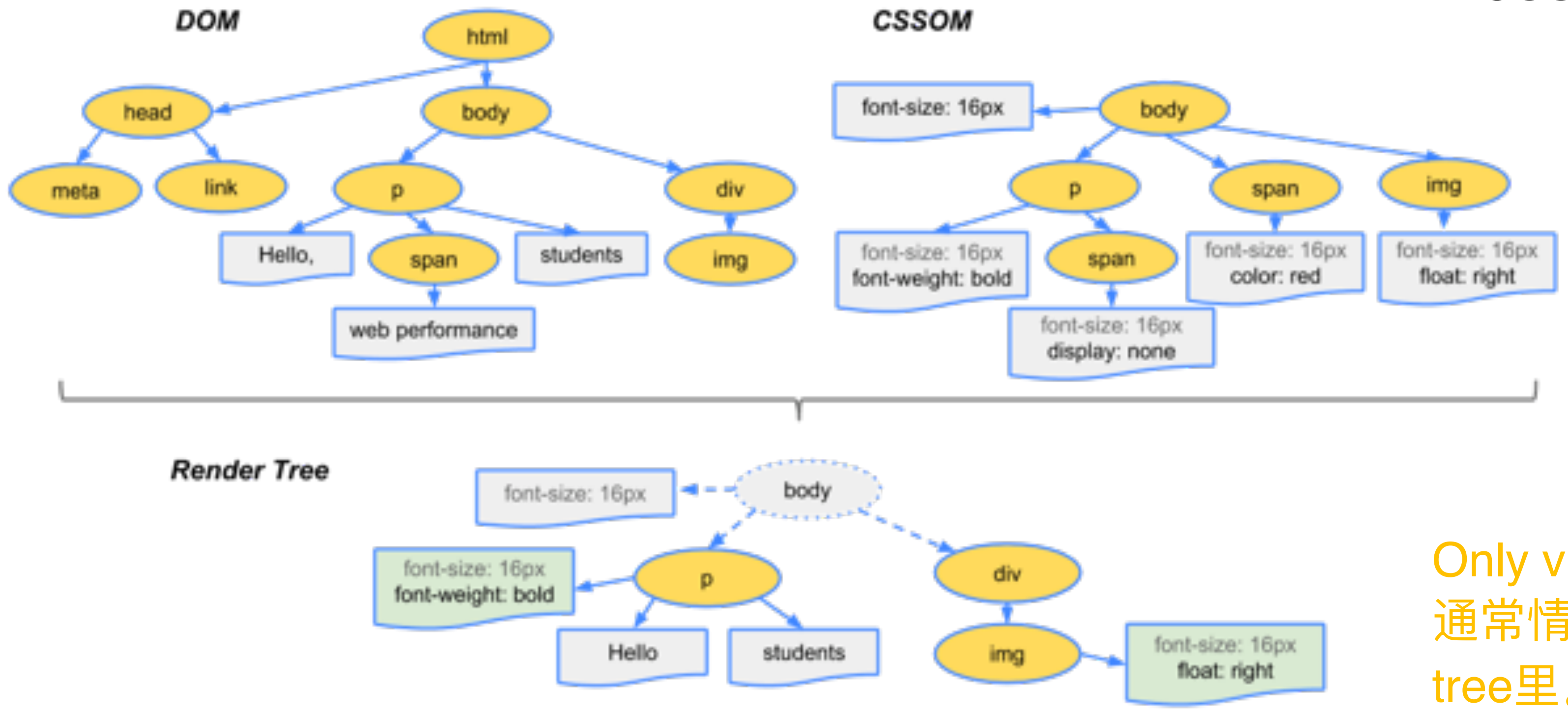
```
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link href="style.css" rel="stylesheet">
    <title>Critical Path: Script</title>
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div><img src="awesome-photo.jpg"></div>
    <script>
      var span = document.getElementsByTagName('span')[0];
      span.textContent = 'interactive'; // change DOM text content
      span.style.display = 'inline';  // change CSSOM property
      // create a new element, style it, and append it to the DOM
      var loadTime = document.createElement('div');
      loadTime.textContent = 'You loaded this page on: ' + new Date();
      loadTime.style.color = 'blue';
      document.body.appendChild(loadTime);
    </script>
  </body>
</html>
```

Minimize CSS rules and remove unused CSS rules.
尽量简化CSS规则，删掉不需要的规则

Render Tree

Dom Tree + CSS = Render Tree

CSS Box Model



Only visual elements are in the render tree.
通常情况下，只有可以被看见的元素会在render tree里。

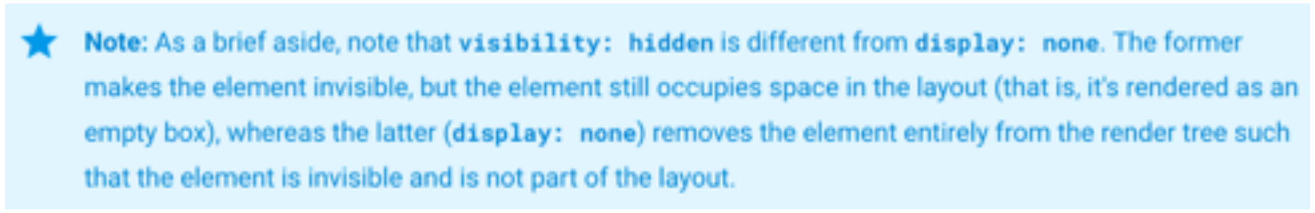
Render Tree

1、每个节点是一个Render Object，包含宽高、位置、背景色等样式信息

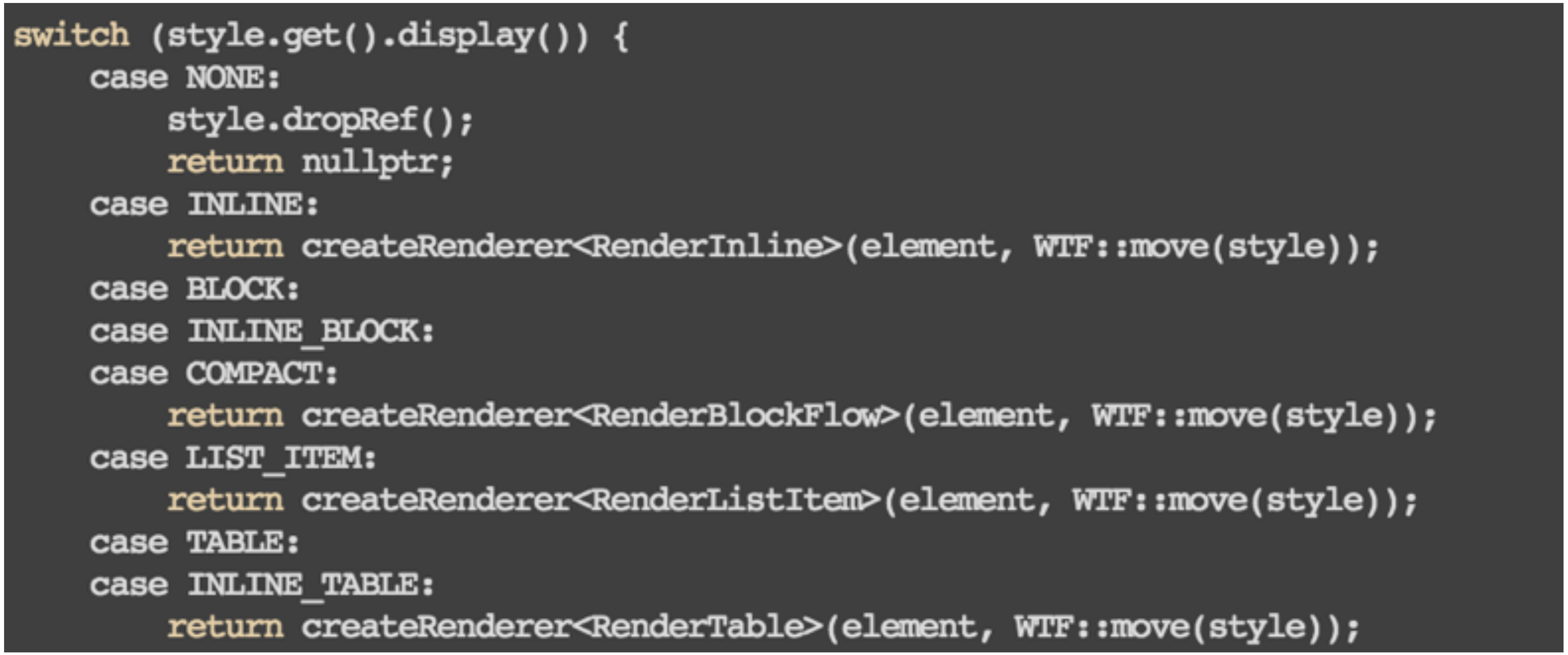
2、Render Tree和DOM Tree不完全对应

3、display: none的元素不在Render Tree中

4、visibility: hidden的元素在Render Tree中


Note: As a brief aside, note that `visibility: hidden` is different from `display: none`. The former makes the element invisible, but the element still occupies space in the layout (that is, it's rendered as an empty box), whereas the latter (`display: none`) removes the element entirely from the render tree such that the element is invisible and is not part of the layout.

Render Tree



```
switch (style.get().display()) {
    case NONE:
        style.dropRef();
        return nullptr;
    case INLINE:
        return createRenderer<RenderInline>(element, WTF::move(style));
    case BLOCK:
    case INLINE_BLOCK:
    case COMPACT:
        return createRenderer<RenderBlockFlow>(element, WTF::move(style));
    case LIST_ITEM:
        return createRenderer<RenderListItem>(element, WTF::move(style));
    case TABLE:
    case INLINE_TABLE:
        return createRenderer<RenderTable>(element, WTF::move(style));
```

domContentLoaded

domLoading: this is the starting timestamp of the entire process, the browser is about to start parsing the first received bytes of the HTML document.

domInteractive: marks the point when the browser has finished parsing all of the HTML and DOM construction is complete.

domContentLoaded: marks the point when both the DOM is ready and there are no stylesheets that are blocking JavaScript execution - meaning we can now (potentially) construct the render tree. 表示DOM解析已经完成，也没有什么样式阻碍JavaScript的执行了。

> Many JavaScript frameworks wait for this event before they start executing their own logic. For this reason the browser captures the EventStart and EventEnd timestamps to allow us to track how long this execution took.
>
> domContentLoaded typically marks when both the DOM and CSSOM are ready.
>
> 通常这个状态标记DOM解析和CSS规则都已准备好。
>
> If there is no parser blocking JavaScript then DOMContentLoaded will fire immediately after domInteractive.

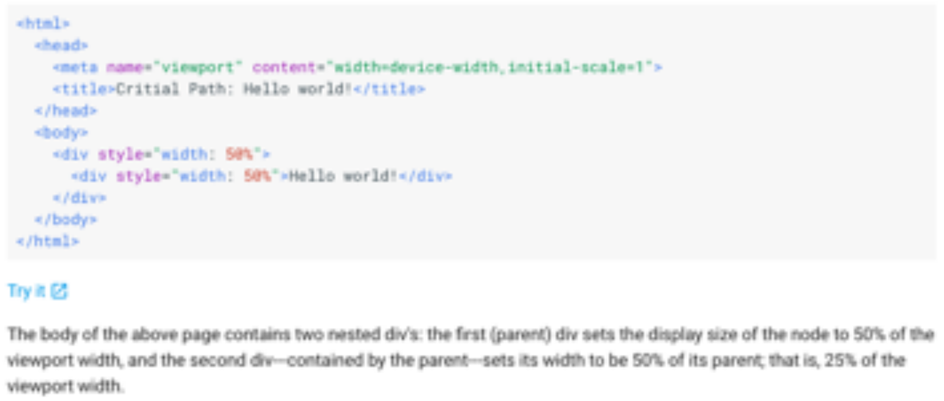
domComplete: as the name implies, all of the processing is complete and all of the resources on the page (images, etc.) have finished downloading - in other words, the loading spinner has stopped spinning.

loadEvent: as a final step in every page load the browser fires an onload event which can trigger additional application logic.

重排(Layout) / 回流(reflow)

页面第一次渲染时，重排指的是：

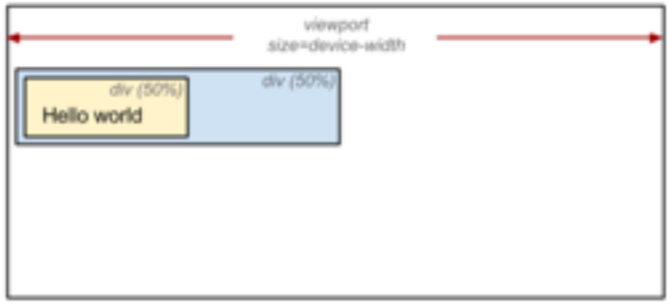
Calculate the position and size 计算对象在浏览器上的位置和大小



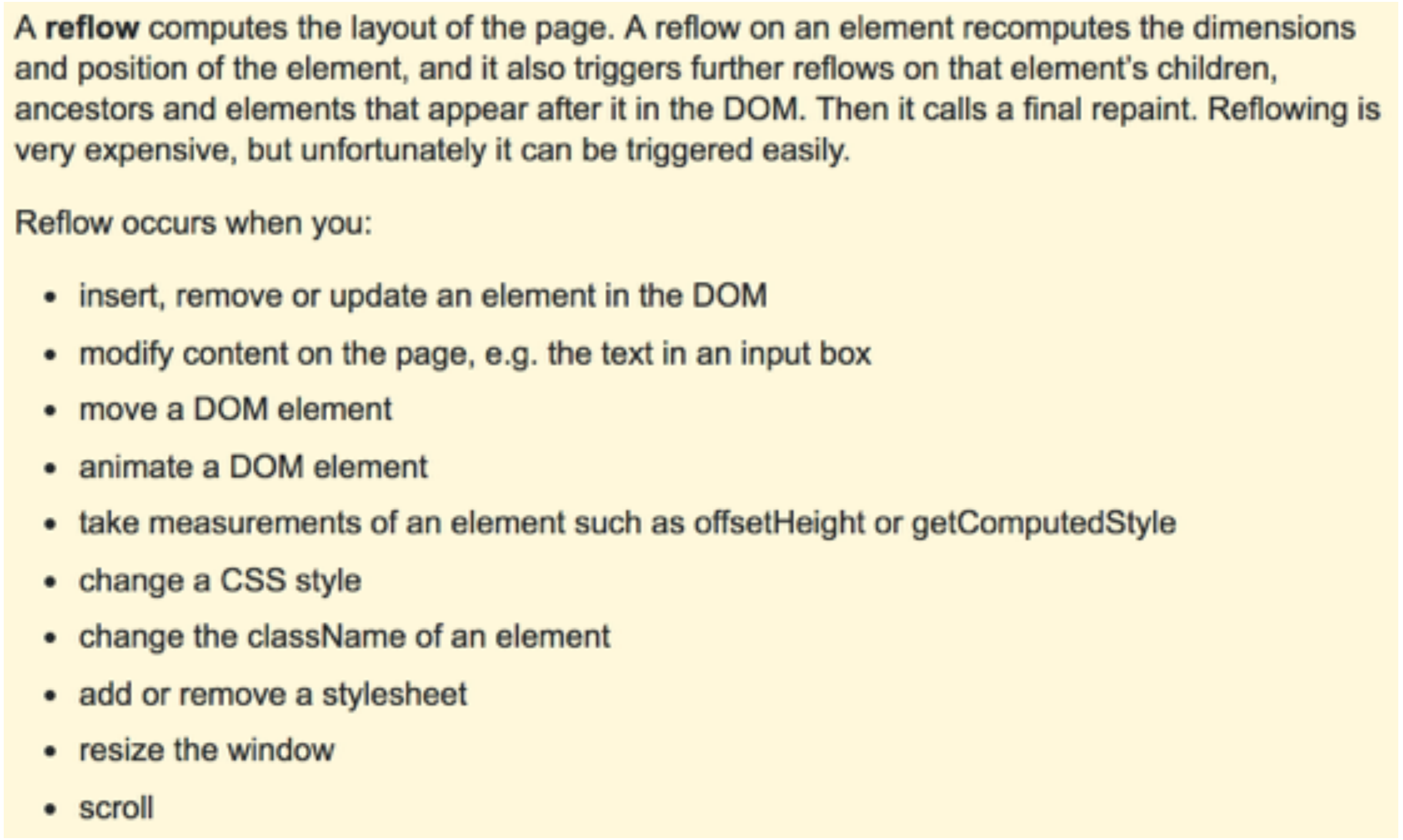
```
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <title>Critial Path: Hello world!</title>
  </head>
  <body>
    <div style="width: 50%">
      <div style="width: 50%">Hello world!</div>
    </div>
  </body>
</html>
```

Try it

The body of the above page contains two nested div's: the first (parent) div sets the display size of the node to 50% of the viewport width, and the second div—contained by the parent—sets its width to be 50% of its parent; that is, 25% of the viewport width.



重排(Layout) / 回流(reflow)



A **reflow** computes the layout of the page. A reflow on an element recomputes the dimensions and position of the element, and it also triggers further reflows on that element's children, ancestors and elements that appear after it in the DOM. Then it calls a final repaint. Reflowing is very expensive, but unfortunately it can be triggered easily.

Reflow occurs when you:

- insert, remove or update an element in the DOM
- modify content on the page, e.g. the text in an input box
- move a DOM element
- animate a DOM element
- take measurements of an element such as offsetHeight or getComputedStyle
- change a CSS style
- change the className of an element
- add or remove a stylesheet
- resize the window
- scroll

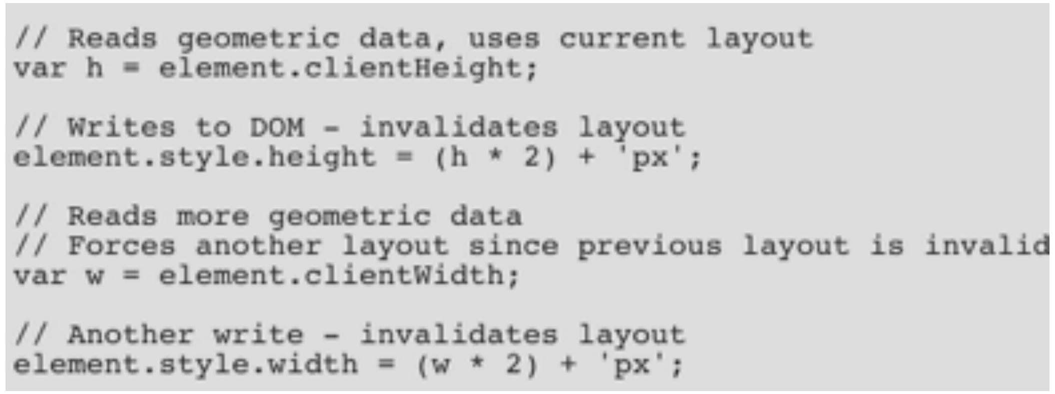
1、当修改元素的位置、大小时，引起浏览器再次重排

2、一个元素的重排可能影响父级元素、子元素和相邻元素

For more info: https://sites.google.com/site/getsnippet/javascript/dom/repaints-and-reflows-manipulating-the-dom-responsibly
https://gist.github.com/paulirish/5d52fb081b3570c81e3a

重排(Layout) / 回流(reflow)

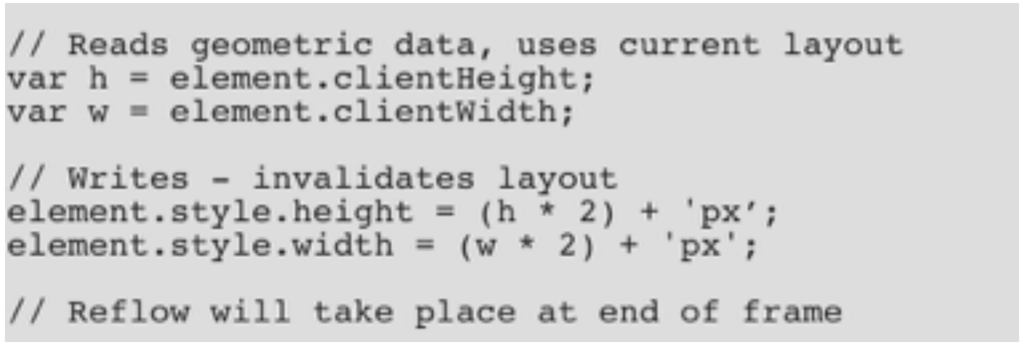
上面提到的js操作会不可避免地引起页面重排，但会等到下一帧再重排；而这些js操作之后的元素几何信息读取会立刻引起重排，非常影响性能。



```
// Reads geometric data, uses current layout
var h = element.clientHeight;

// Writes to DOM - invalidates layout
element.style.height = (h * 2) + 'px';

// Reads more geometric data
// Forces another layout since previous layout is invalid
var w = element.clientWidth;

// Another write - invalidates layout
element.style.width = (w * 2) + 'px';
```

Most devices today refresh their screens 60 times a second.
也就是所谓的 60fps = 60 frames per second, 60帧每秒

最好这样做



```
// Reads geometric data, uses current layout
var h = element.clientHeight;
var w = element.clientWidth;

// Writes - invalidates layout
element.style.height = (h * 2) + 'px';
element.style.width = (w * 2) + 'px';

// Reflow will take place at end of frame
```

Whenever the DOM is written to, the current layout is ‘invalidated,’ and will need to be reflowed. The browser usually waits to do this until the end of the current operation or frame.
However, if we ask (via JavaScript) for geometric values before the current operation or frame is complete, the browser is forced to reflow the layout immediately. This is known as a ‘forced synchronous layout,’ and has the potential to be devastating to performance.

For more info: https://engineering.gosquared.com/optimising-60fps-everywhere-in-javascript

重排(Layout) / 回流(reflow)

如何避免重排?

1、用transform做形变和位移

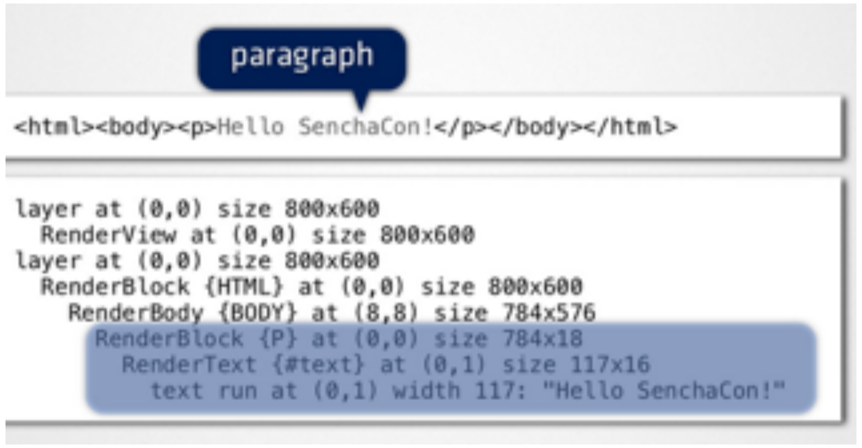
2、通过绝对定位、固定定位，脱离当前层叠上下文（即形成新的Render Layer)

3、减少不必要的DOM深度 Reduce unnecessary DOM depth

4、尽量避免使用table布局   Avoid table layouts, they trigger more reflows than block layouts because multiple passes must be made over the elements. 使用table-layout: fixed属性。

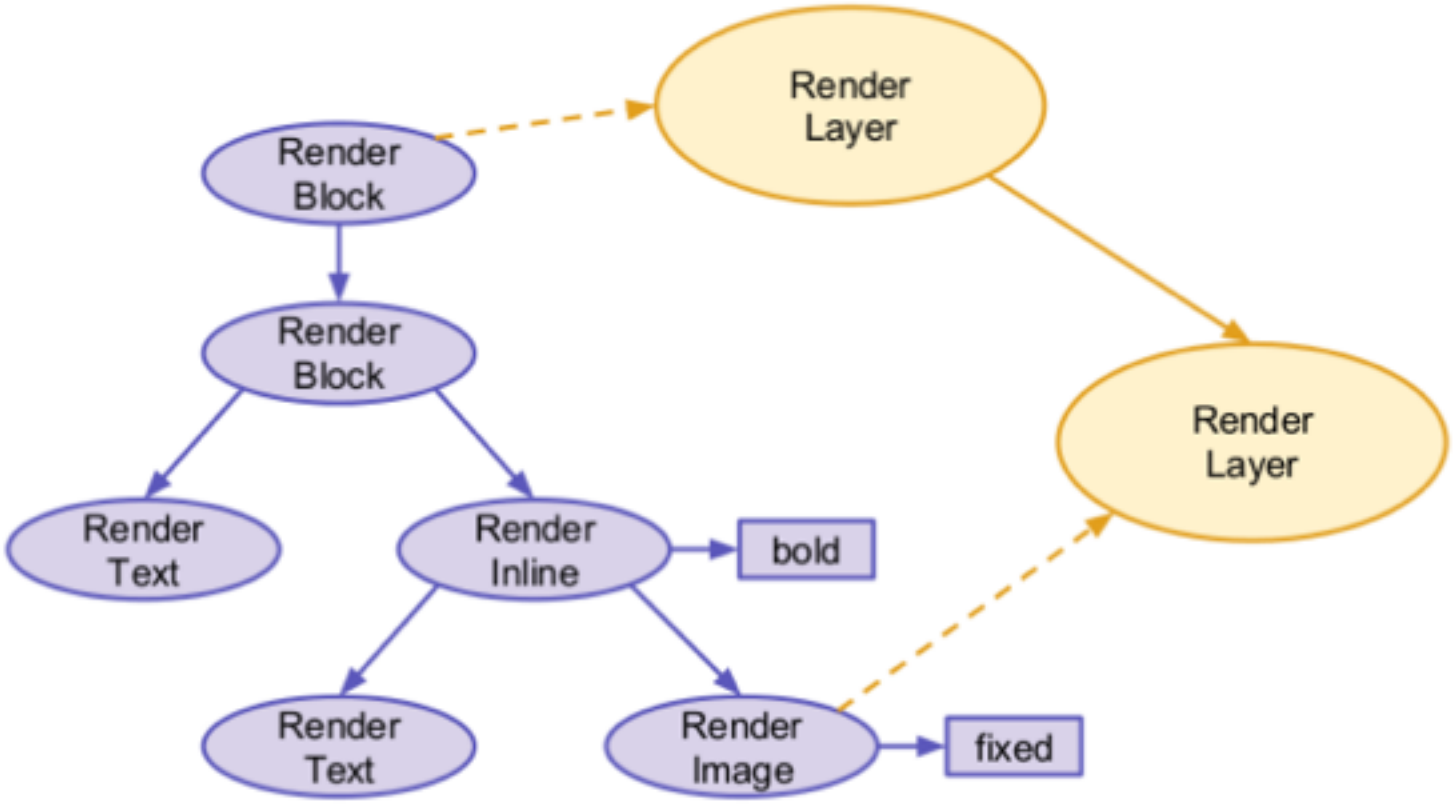
生成Render Layer

During construction or modification of the render tree,
some renderers will receive associated
WebCore::RenderLayer objects.
Attaching a RenderLayer to a RenderObject indicates
that the renderer has its own coordinate system,
and therefore the content of the renderer
and its descendants can (but won't necessarily) be
painted to a separate compositing layer.
Each RenderLayer is added to the RenderLayer tree,
which has a sparse association with the render tree and is
the primary data structure used for the painting phase of rendering.



```
<html><body><p>Hello SenchaCon!</p></body></html>
```

```
layer at (0,0) size 800x600
  RenderView at (0,0) size 800x600
layer at (0,0) size 800x600
  RenderBlock {HTML} at (0,0) size 800x600
    RenderBody {BODY} at (8,8) size 784x576
      RenderBlock {P} at (0,0) size 784x18
        RenderText {#text} at (0,1) size 117x16
          text run at (0,1) width 117: "Hello SenchaCon!"
```



生成Graphics Layer

将Layer Tree上的某些节点进一步提升与合并

1、video、canvas元素，flash插件

2、拥有perspective、CSS3D变形的元素

3、backface-visibility为hidden

4、对opacity、transform、filter、backdropfilter应用了animation或者transition

5、设置了will-change属性的元素

GraphicsLayers are what get uploaded to the GPU as textures.

What's a texture?

Think of it as a bitmap image that's moved from main memory (i.e. RAM) to video memory (i.e. VRAM, on your GPU).

生成Graphics Layer

优势：

1、GPU直接渲染，快于CPU

劣势：

1、过多的合成层会造成GPU传输的压力

2、每一个合成层都会有其对应的内存对象，过多的合成层会占用过多的内存资源

总结

Performance is the art of avoiding work, and making any work you do as efficient as possible.

性能是有关“避免做某些工作”以及让你做的工作尽可能的高效的艺术。

THANKS

谢谢您的观看