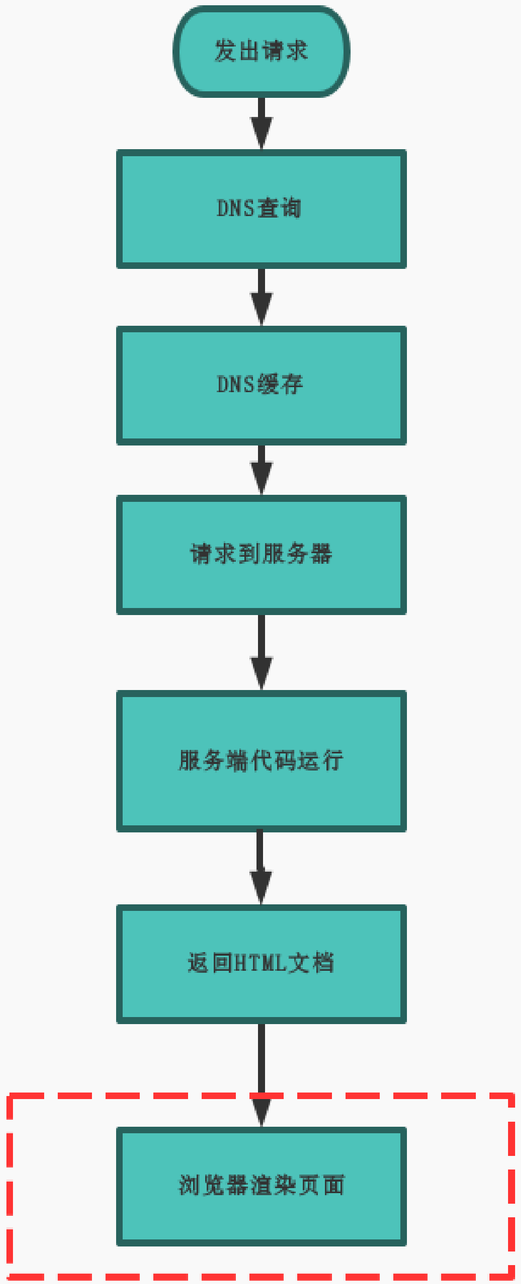
浏览器渲染过程与原理（上）

浏览器渲染过程与原理（上）

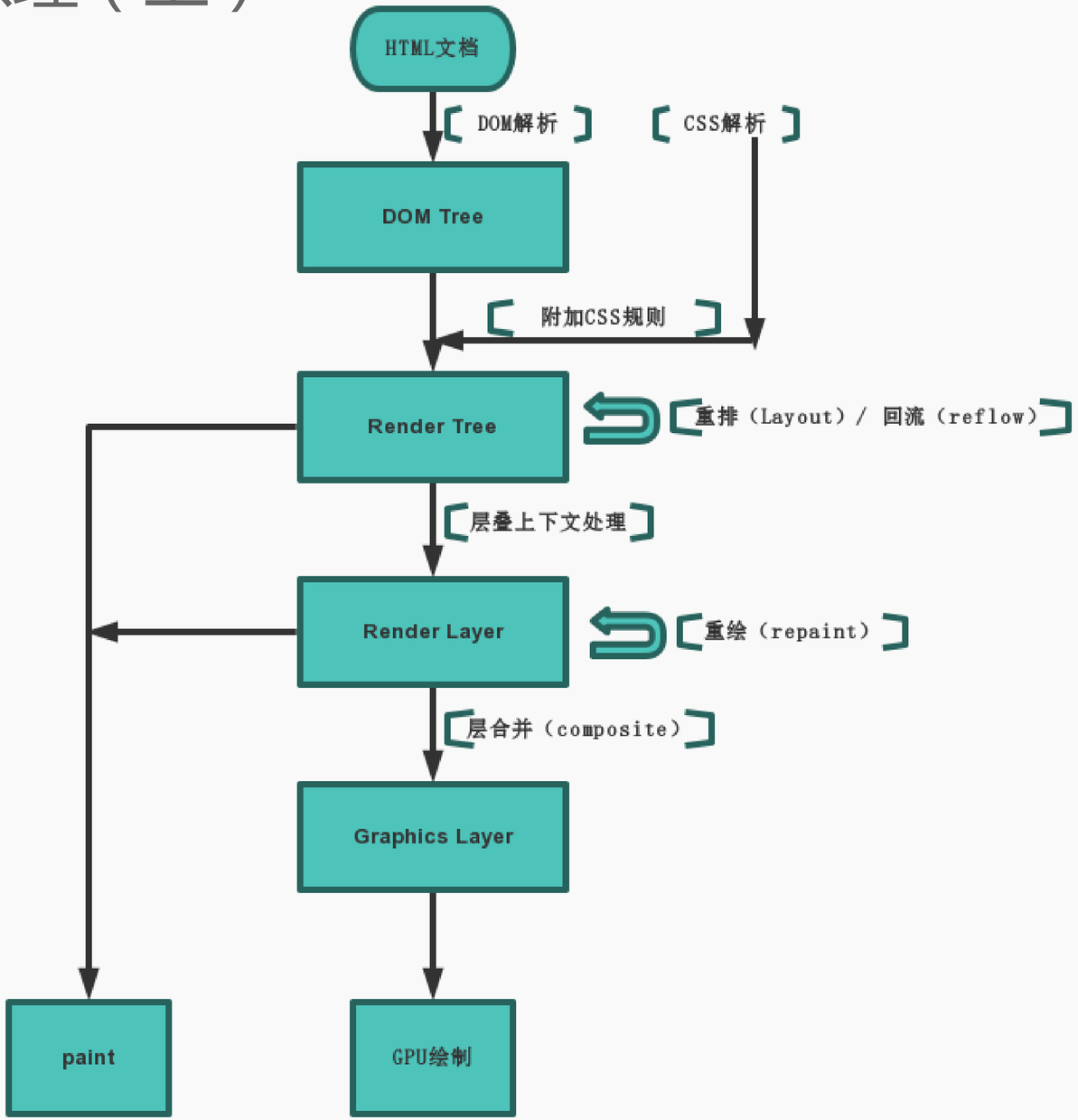


浏览器渲染过程与原理（上）

浏览器渲染引擎

- IE（Trident）
- Chrome（Blink）
- Firefox（Gecko）
- Opera（Blink）
- Safari（Webkit）
- UC（U3）
- QQ浏览器/微信webview（X5/Blink）

浏览器渲染过程与原理（上）



浏览器渲染过程与原理（上）

DOM解析

把HTML文档解析为DOM树的过程

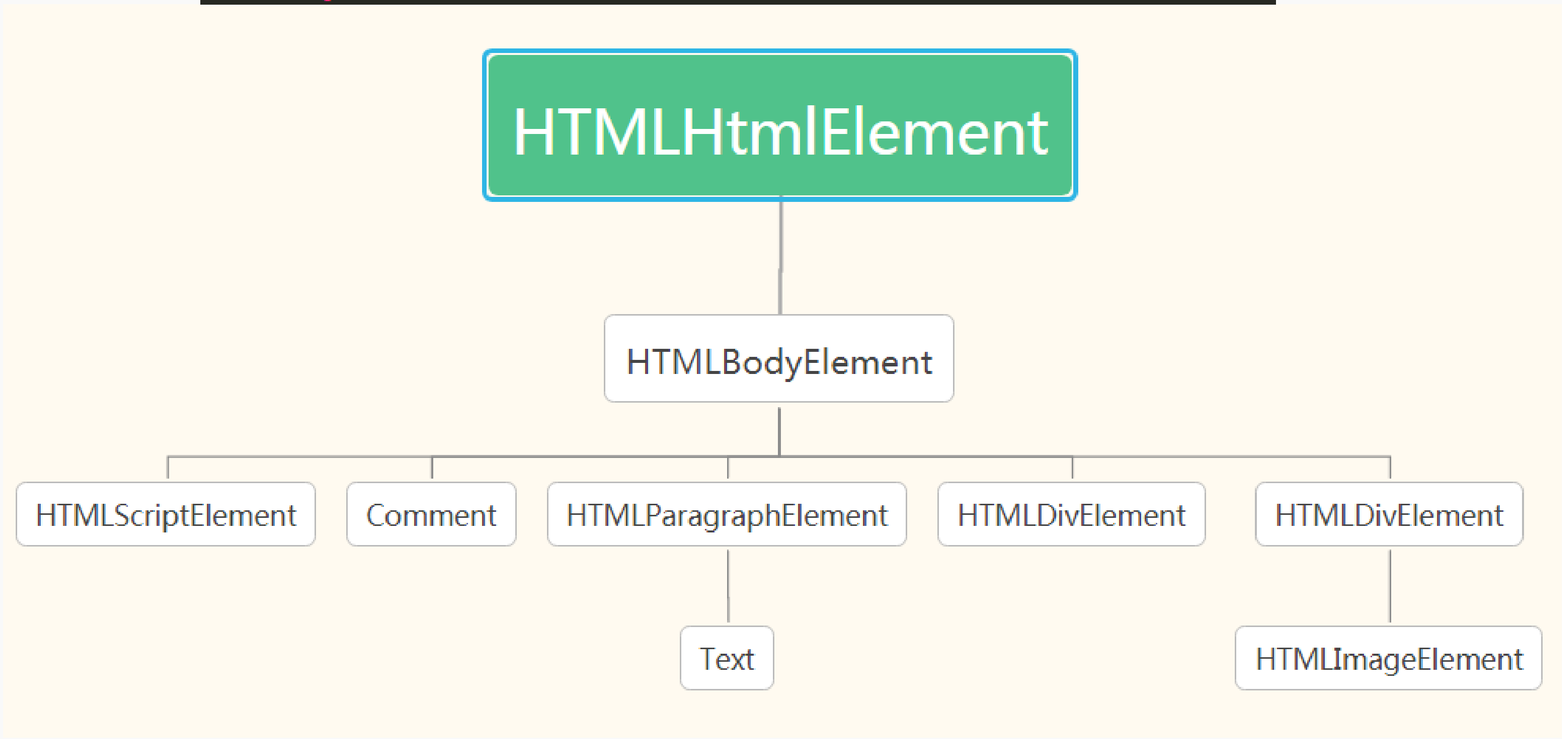
- 遇到<scirpt>标签则停止解析，先执行js
- DOM解析完成后触发DOMContentLoaded事件
- 此时图片资源并未加载完成

浏览器渲染过程与原理（上）

DOM Tree

```
<body>
    <script></script>
    <!--这是注释-->
    <p>DOM解析</p>
    <div id="d1"></div>
    <div>
        <img src="https://cdn.pixabay.com/photo/
        2418339_960_720.jpg">
    </div>

</body>
```



浏览器渲染过程与原理（上）

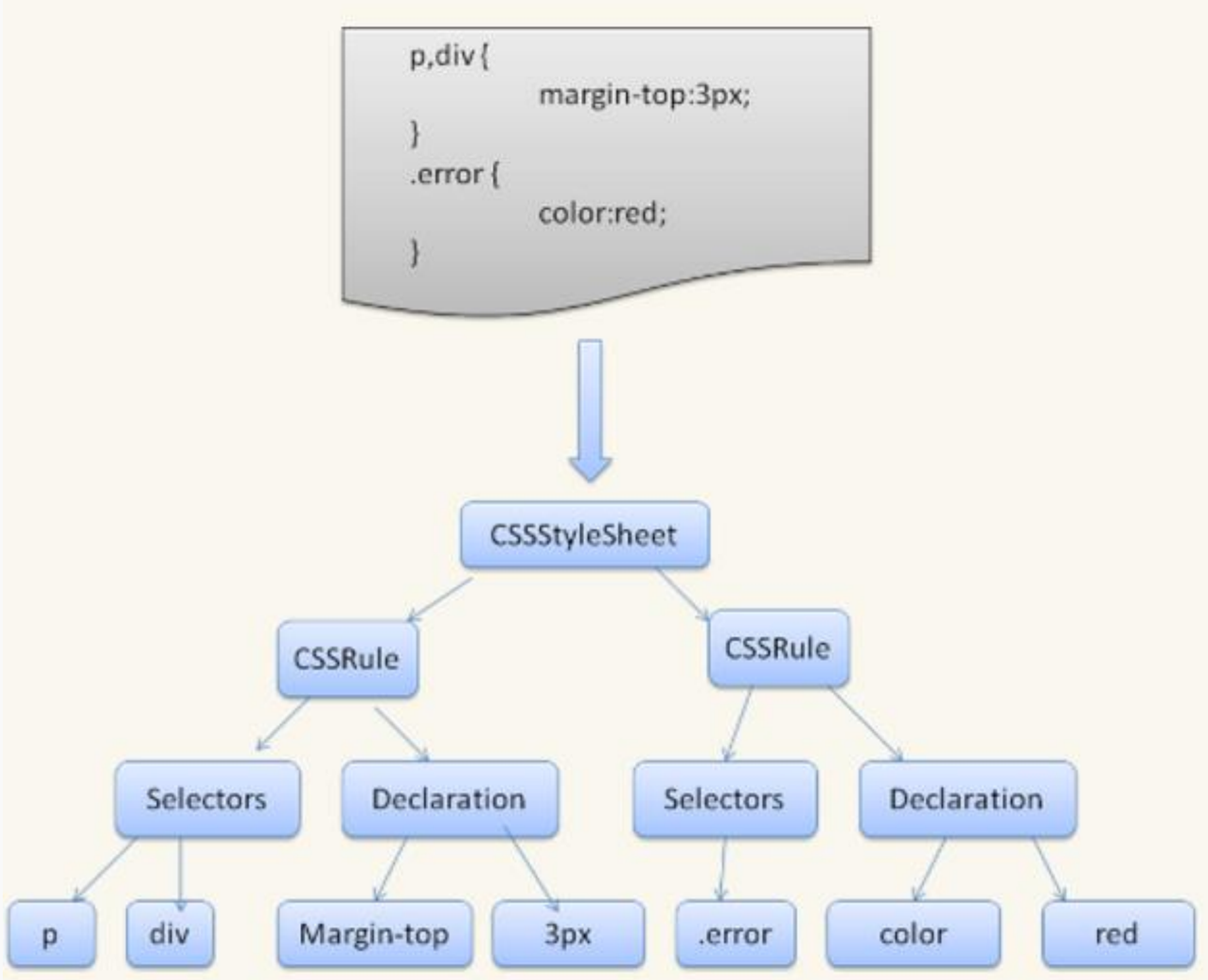
DOM Tree

DOM树结构与HTML标签一一对应

- display: none的元素也在DOM树中
- <scirpt>标签也在DOM树中
- 注释也在DOM树中

浏览器渲染过程与原理（上）

CSS解析



浏览器渲染过程与原理（上）

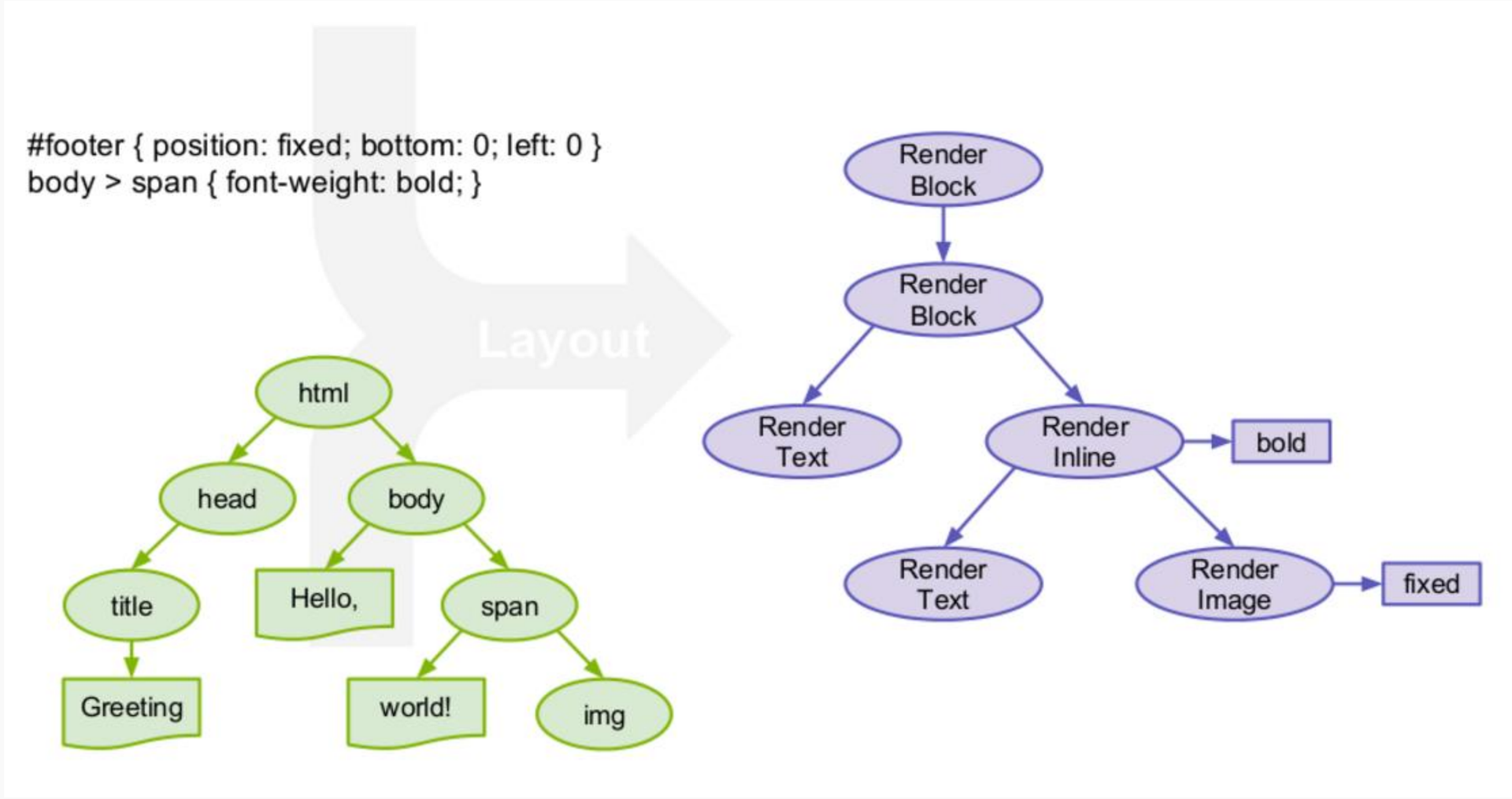
CSS解析

将CSS代码解析为CSS规则树的过程

- 与DOM解析同步进行
- 与script的执行互斥
- Webkit内核进行了script执行优化

浏览器渲染过程与原理（上）

Render Tree



浏览器渲染过程与原理（上）

Render Tree

DOM Tree + CSS Rules = Render Tree

- 每个节点为一个Render Object对象，包含宽高、位置、背景色等样式信息
- 宽高和位置是通过Layout（重排）计算出
- Render Tree和DOM Tree不完全对应
- display: none的元素不在Render Tree中
- visibility: hidden的元素在Render Tree中
- float元素、absolute元素、fixed元素会发生位置偏移
- 常说的脱离文档流，就是脱离Render Tree

浏览器渲染过程与原理（上）

重排（Layout）/回流（reflow）

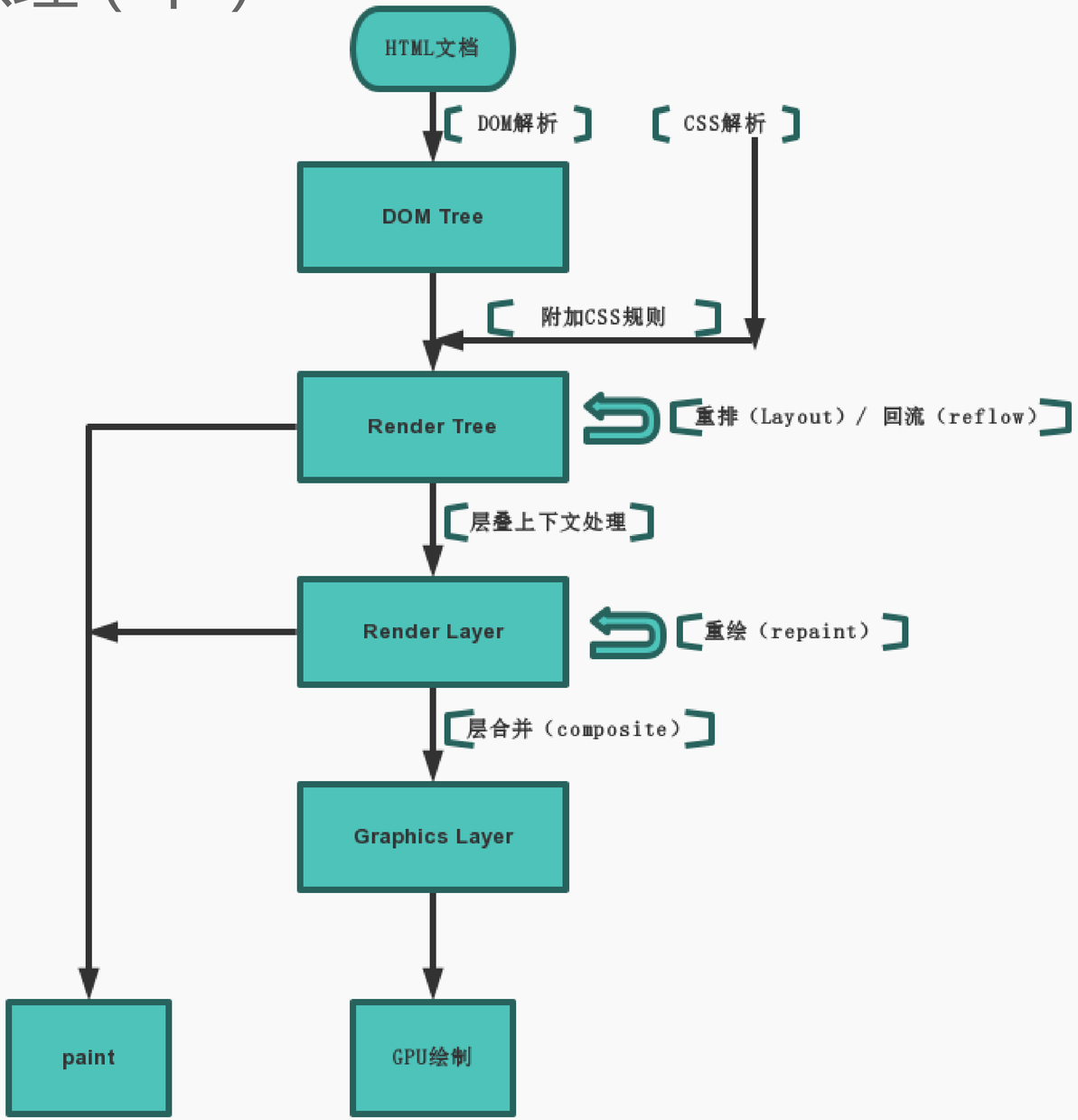
- 当修改元素的位置、大小时，引起浏览器的重排
- 对一个元素的重排，可能影响到其父级元素和相邻元素

如何避免重排？

- 用transform做形变和位移
- 通过绝对定位，脱离当前层叠上下文（即形成新的Render Layer）

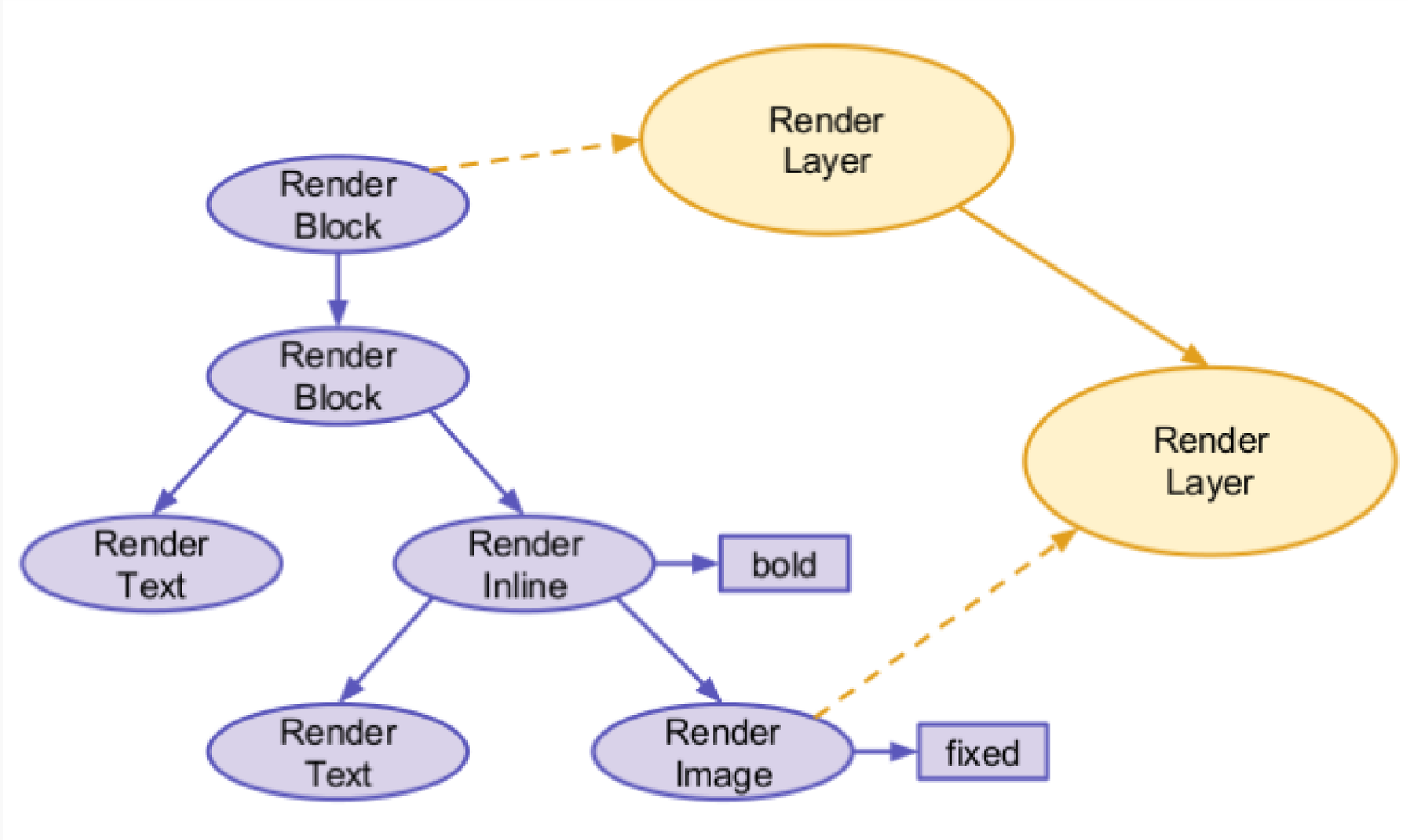
浏览器渲染过程与原理（下）

浏览器渲染过程与原理（下）



浏览器渲染过程与原理（下）

生成Render Layer



浏览器渲染过程与原理（下）

生成Render Layer

将Render Tree上的某些节点提升到同一个Layer的过程

- 处理定位、裁剪、页内滚动、CSS Transform/Opacity/Animation/Filter、z-index排序等
- 所有Render Layer组合成一棵Layer Tree
- 浏览器基于Layer Tree进行Paint

浏览器渲染过程与原理（下）

生成Render Layer的条件

- 根元素（HTML）
- 有明确的定位属性（relative、fixed、sticky、absolute）
- 透明的（opacity 小于 1）
- 有 CSS 滤镜（filter）
- 有 CSS mask 属性
- 有 CSS mix-blend-mode 属性（不为 normal）
- 有 CSS transform 属性（不为 none）

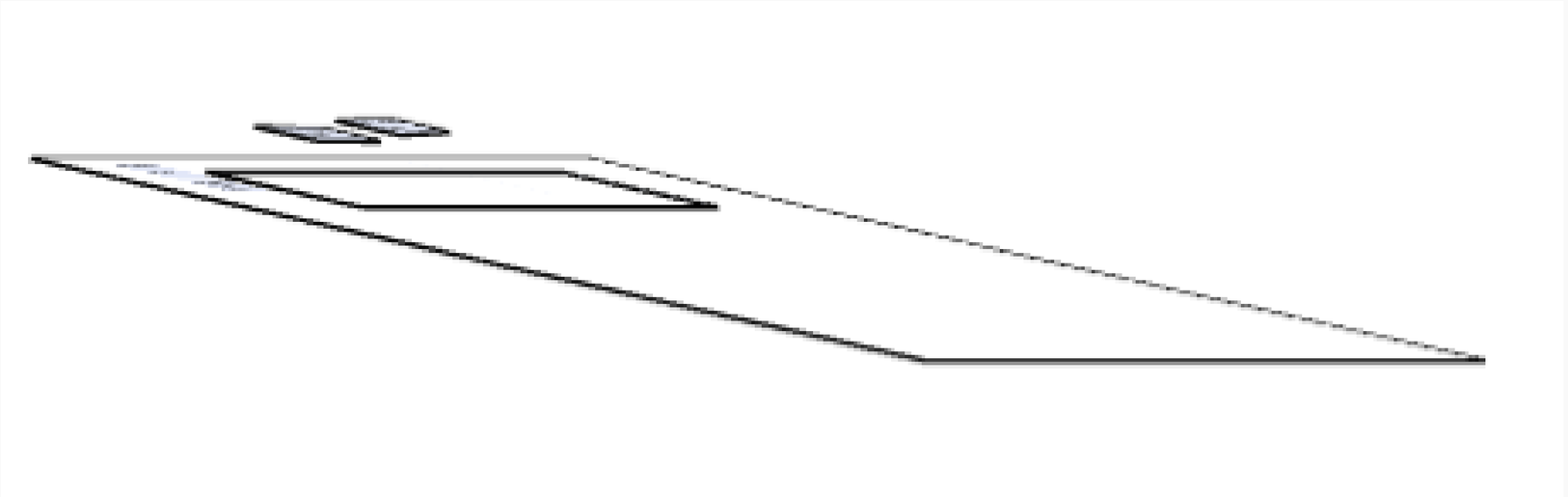
浏览器渲染过程与原理（下）

生成Render Layer的条件

- backface-visibility 属性为 hidden
- 有 CSS reflection 属性
- 有 CSS column-count 属性（不为 auto）或者 有 CSS column-width 属性（不为 auto）
- 当前有对于 opacity、transform、filter、backdrop-filter 应用动画
- overflow 不为 visible
- 不需要 paint 的 PaintLayer，比如一个没有视觉属性（背景、颜色、阴影等）的空 div

浏览器渲染过程与原理（下）

生成Graphics Layer



浏览器渲染过程与原理（下）

生成Graphics Layer

将Layer Tree上的某些节点进一步提升与合并

优势：

- GPU直接渲染，快于CPU
- 当需要 repaint 时，只需要 repaint 本身，不会影响到其他的层
- 对于 transform 和 opacity 效果，不会触发 layout 和 paint

浏览器渲染过程与原理（下）

生成Graphics Layer

- video、canvas元素，flash插件
- 拥有perspective、CSS3D变形的元素
- backface-visibility 为 hidden
- 对 opacity、transform、filter、backdropfilter 应用了 animation 或者 transition
- 设置了will-change属性的元素
- 层之间的层叠遮盖

浏览器渲染过程与原理（下）

- 没有Graphics Layer的元素与父元素共属同一个
- 过多的合成层会造成GPU传输的压力

基本的前端优化手段（上）

基本的前端优化手段（上）

- 资源文件的引入位置
- 异步script标签
- 避免使用css@import
- 注意空的src和href

基本的前端优化手段（上）

资源引入位置

原则：

- 尽可能快的展示出页面内容
- 尽可能快的使功能可用

知识点：

- 文档从上到下依次解析
- 解析css的时候会阻塞js的执行
- 解析js的时候会阻塞css的解析和页面的渲染

基本的前端优化手段（上）

资源引入位置

- css文件放在head中，先外链的，后本页的
- js文件放在body底部，先外部库，后本页的
- 兼容处理的js文件应放在head中，如babel-polyfill.js
- 页面布局的js文件应放在head中，如flexible.js
- body中间尽量不写style标签和script标签

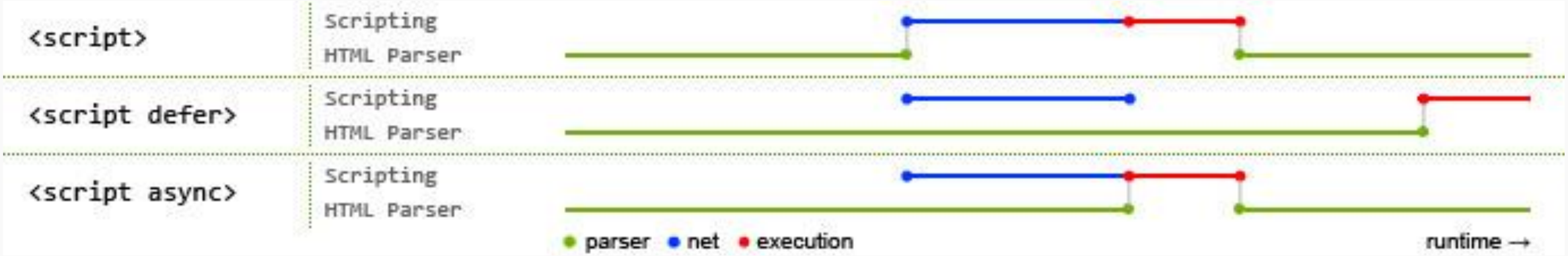
基本的前端优化手段（上）

异步加载js文件

资源需要在头部、body中间加载的时候怎么办？

defer：异步加载，最快在DOMContentLoaded事件之前执行

async：异步加载，加载完毕立即执行



基本的前端优化手段（上）

异步加载js文件

使用建议：

- async执行时机不确定，不建议用于业务代码
- async可用于单独的代码，如第三方统计代码
- defer的实际效果与将代码放在body底部一样
- 多个加了defer的js文件也可能出现执行顺序错乱的现象

基本的前端优化手段（上）

避免使用css@import

css的@import造成额外的请求

基本的前端优化手段（上）

使用sass中的@import

- sass中的@import会将css文件进行合并
- 多次@import同一个文件会造成代码重复
- sass中使用@import引用“抽象”的内容，如：变量、占位符、函数

基本的前端优化手段（上）

避免在页面出现空的href和src

- ~~img标签给空的src会请求向当前页面地址~~
- ~~script标签给空的src会请求想当前页面地址~~
- a标签给空的href，会重定向到当前页面地址
- Form给空的method，会提交表单到当前页面地址

基本的前端优化手段（下）

基本的前端优化手段（下）

- 缓存DOM
- 批量操作DOM
- 在内存中操作DOM
- DOM读写分离
- 事件代理
- 最小化全局影响

基本的前端优化手段（下）

缓存DOM

方法：

```
var div = document.getElementsByTagName('div');
var container2 = document.querySelectorAll('.container2');
```

原因：

- 同一个节点，无需多次查询
- 查询DOM耗时

基本的前端优化手段（下）

缓存DOM

HTMLCollection 与NodeList的区别：

- HTMLCollection是动态的，页面节点发生变化后，引用随之更新
- 通过querySelecorAll取到的NodeList为静态的

所以缓存为哪种类型需要额外注意！

基本的前端优化手段（下）

批量操作DOM

方法：

先用字符串拼接完毕，再用innerHTML更新DOM

原因：

- 避免频繁操作DOM（DOM操作是耗时的）
- 避免发生重复渲染

基本的前端优化手段（下）

在内存中操作DOM

方法：

使用DocumentFragment对象

原因：

- 让DOM操作发生在内存中，而不是页面上
- 避免频繁访问DOM

基本的前端优化手段（下）

DOM读写分离

方法：

修改DOM动作与访问DOM分开批量进行

原因：

- 浏览器的“惰性渲染”机制
- 读取DOM会触发浏览器的一次渲染

基本的前端优化手段（下）

使用事件代理

方法：

将事件监听器注册在父级元素

原因：

- 减少内存占用
- 能捕获到动态添加的节点事件

基本的前端优化手段（下）

最小化全局影响

- 清除对象引用
- 清除定时器
- 清除事件监听器
- 创建最小作用域变量（及时回收）