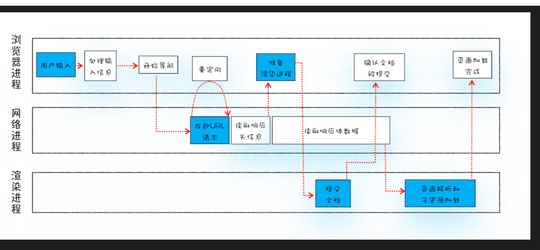


- 用户输入

1. 用户输入url后并回车
2. 浏览器进程会对url进行处理，如果是符合url规则，组装协议，否则使用默认搜索引擎。
3. 把url通过IPC提交给网络进程。

- url请求

1. 网络进程会判断本地有没有缓存资源，有则返回缓存资源（具体参考缓存策略）。
2. 否则网络进程则发起http请求
   1. 进行DNS解析，获取ip地址，默认是80端口（https:443端口）。
   2. 与目标服务区建立tcp协议，在建立http协议。（chrome限制6个tcp链接，若果超过请排队，同一域名的情况下。）
   3. 发送请求行，请求头，请求体。
   4. 拿到服务器响应行，响应头，响应体。
      4.1 服务服务器返回的code是30x，浏览器则需要发起重定向，根据请求头里面的Location字段。
      4.2 在检查响应头Content-Type的类型，如果是字节流类型，发起下载，是html的则 **通知浏览器进程准备**渲染进程。

- 准备渲染进程

浏览器进程会根据url是不是已存在的渲染进程，同处于一个站点，相同站台就利用一个渲染进程。否则新开一个渲染进程。

- 提交文档
  1. **渲染进程**会发送“提交文档”给浏览器进程。
  2. 浏览器进程收到后，会开始清理之前的文档，发出“确认提交”给渲染进程。同时浏览器进程会更新浏览器界面状态，包括了安装状态，地址的url，前进推历史。
  3. 渲染进程再收到“确认提交”后，会与网络进程建立传输管道，便开始执行解析数据、下载子资源等后续流程，并实时向浏览器进程更新最新的渲染状态。
- 渲染阶段

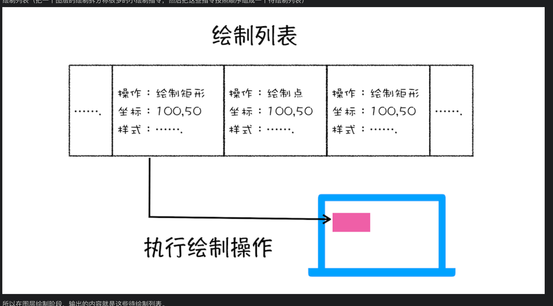
构建Dom树
样式计算

1. 把css文本转化为浏览器可以理解的stylesheet结构。
2. 然后使其样式表里面的值标准化（em,red）。
3. 计算出Dom中的具体样式（继承之类的）。

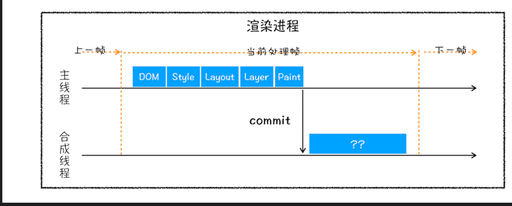
布局阶段

1. 创建布局树
   遍历Dom树中可见的节点，并把这些节点加到布局中。
2. 计算布局(复杂todo)

分层
布局树->图层树（拥有层叠上下文属性的元素被单独提为一层）
图层绘制
绘制列表（把一个图层的绘制拆分称很多的小绘制指令，然后把这些指令按照顺序组成一个待绘制列表）

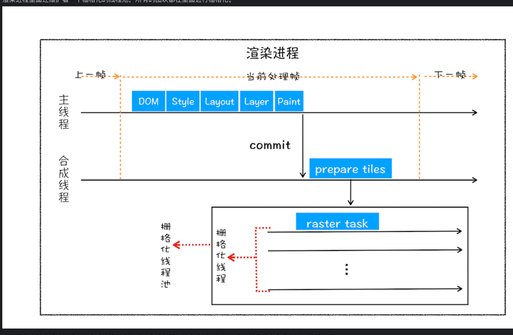


所以在图层绘制阶段，输出的内容就是这些待绘制列表。
栅格化（raster）操作
绘制列表实际上只是用来记录绘制顺序和绘制指令，而实际上绘制操作是由渲染进程的合成线程来完成的。



如图所示，当图层的绘制列表准备完之后，主线程会把绘制列表提交（commit）给渲染进程的合并线程。
视口viewpoint，值得屏幕上可视区域。有些情况下，比如有的页面滚动的内容很长，这个时候如果把整个图层raster，那将会很浪费资源，造成很大的开销。

- 基于这个原因，合成线程会见图层划分为图块（tile），大小通常为256256、512512。
- 合成线程会将感试图附近的图块优先生成为位图，实际生成为图是由栅格化来执行的。栅格化就是把图块变成位图，而图块就是栅格化的最小单位，那栅格化在哪里进行呢？
- 渲染进程里面进维护着一个栅格化的线程池。所有的图块都在里面进行栅格化。



- 一旦所有的图块都被光栅化，合成线程就会生成一个绘制图块的命令“DrawQuad”，然后将命令提交给浏览器进程。

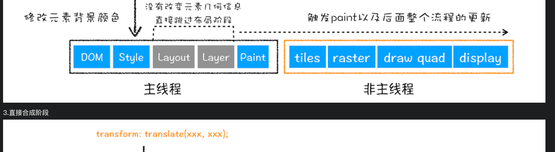
总结：

1. 渲染进程将 HTML 内容转换为能够读懂的 DOM 树结构。
2. 渲染引擎将 CSS 样式表转化为浏览器可以理解的 styleSheets，计算出 DOM 节点的样式。（样式标准化）
3. 创建布局树，并计算元素的布局信息。
4. 对布局树进行分层，生成图层树。
5. 为每个图层生成绘制列表，并提交到合成线程。
6. 合成进程将图层分块，然后在光栅化进程池合成位图。
7. 合成线程发送绘制图块命令 DrawQuad 给浏览器进程。
8. 浏览器进程根据DrawQuad生成页面。

相关概念

1.更新了元素的几何属性（重排）

2.更新元素的绘制属性（重绘）



3.直接合成阶段

这样的效率是最高的，因为是在非主线程上合成，并没有占用主线程的资源，另外也避开了布局和绘制两个子阶段，所以相对于重绘和重排，合成能大大提升绘制效率。