浏览器原理1

Chrome架构

进程和线程

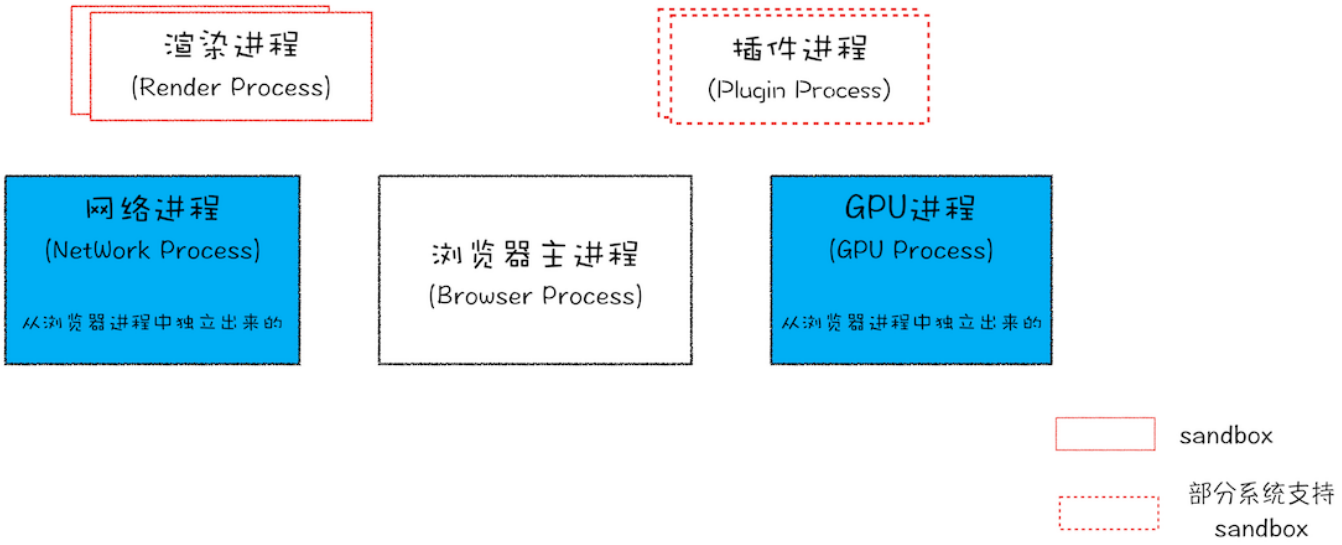
- 进程是资源分配的最小单位，线程是程序执行的最小单位（资源调度的最小单位）
- 进程有自己的独立地址空间，每启动一个进程，系统就会为它分配地址空间，建立数据表来维护代码段、堆栈段和数据段，这种操作非常昂贵。
- 而线程是共享进程中的数据的，使用相同的地址空间，因此CPU切换一个线程的花费远比进程要小很多，同时创建一个线程的开销也比进程要小很多。
- 线程之间的通信更方便，同一进程下的线程共享全局变量、静态变量等数据，而进程之间的通信需要以通信的方式（IPC)进行。不过如何处理好同步与互斥是编写多线程程序的难点。
- 但是多进程程序更健壮，多线程程序只要有一个线程死掉，整个进程也死掉了，而一个进程死掉并不会对另外一个进程造成影响，因为进程有自己独立的地址空间。

浏览器时代

- 单进程：不稳定、不流畅、不安全

多进程浏览器时代

- 浏览器进程：负责用户交互、子进程管理和文件储存等功能
- 渲染进程：从网络下载的 HTML、JavaScript、CSS、图片等资源解析为可以显示和交互的页面（沙箱）
- 网络进程：面向渲染进程和浏览器进程等提供网络下载功能
- GPU 进程
- 插件进程



未来的架构

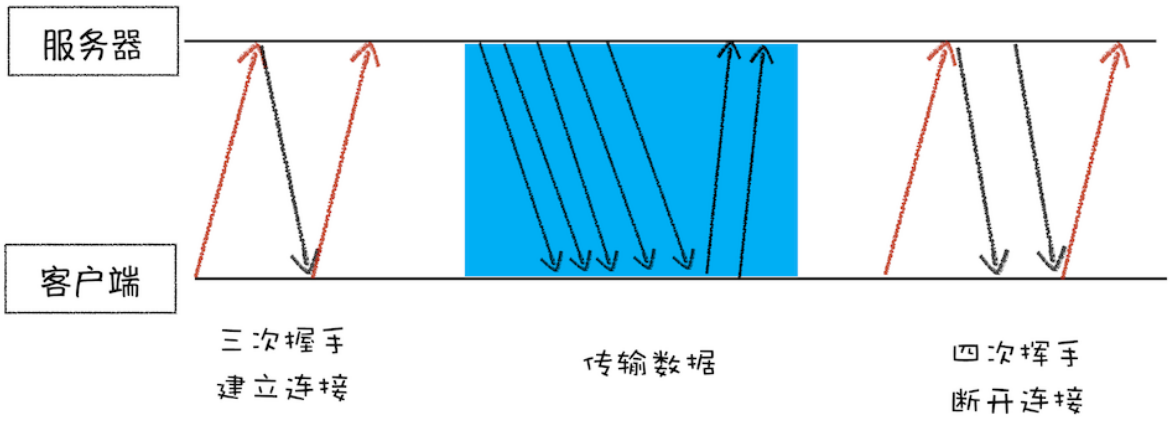
- 面向服务的架构（SOA）

前端网络

重要性：性能

TCP/IP

- 计算机的地址就称为 IP 地址，访问任何网站实际上只是你的计算机向另外一台计算机请求信息
- IP：把数据包送达目的主机
- UDP：把数据包送达应用程序,不能保证数据可靠性，但是传输速度却非常快
- TCP：把数据完整地送达应用程序,是一种面向连接的、可靠的、基于字节流的传输层通信协议



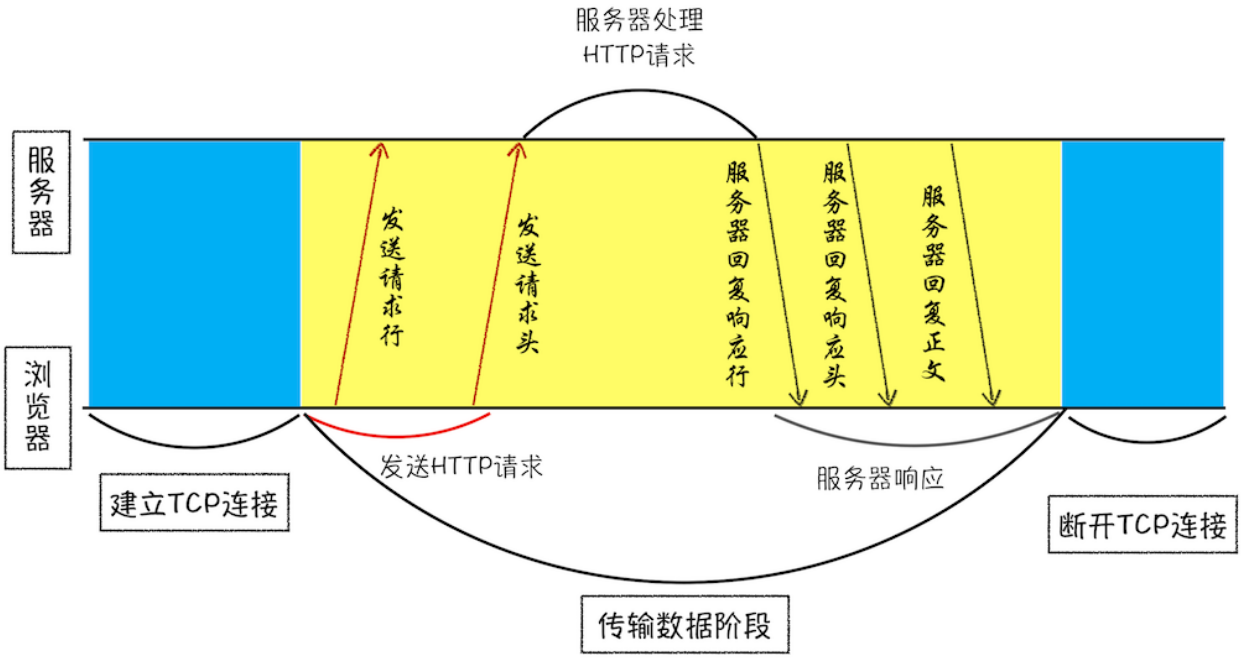
HTTP请求流程

HTTP 协议，正是建立在 TCP 连接基础之上的。HTTP 是一种允许浏览器向服务器获取资源的协议，是 Web 的基础,是浏览器使用最广的协议

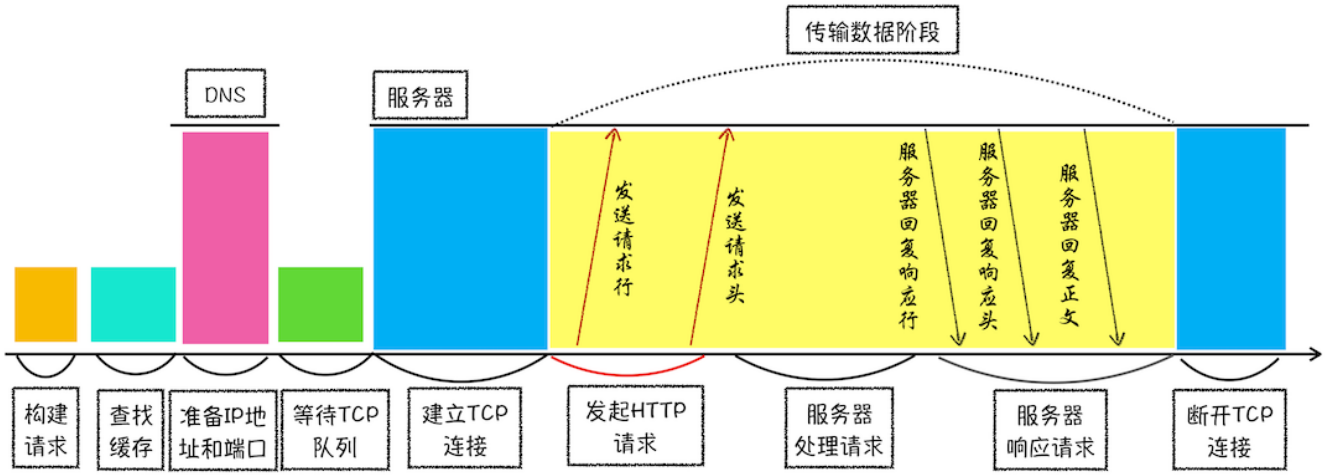
1. 构建请求

```
```

2. 查找缓存:强缓存和协商缓存（304）
3. 准备 IP 地址和端口：DNS



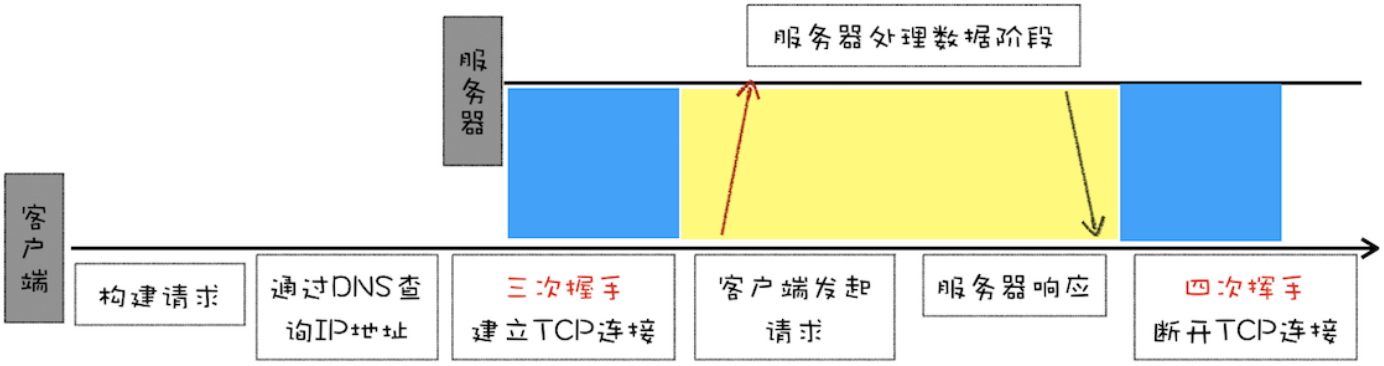
4. 等待 TCP 队列:同一个域名同时最多只能建立 6 个 TCP 连接
5. 建立 TCP 连接
6. 发送 HTTP 请求:请求行、请求头、请求体
7. 服务器端处理 HTTP 请求流程后返回请求：响应行（状态吗）、响应头、响应体
8. 断开连接（Keep-Alive）



HTTP简史及未来

超文本传输协议 HTTP/0.9

- 只有一个请求行，并没有 HTTP 请求头和请求体
- 服务器没有返回头信息，服务器端并不需要告诉客户端太多信息，只需要返回数据就可以了
- 因为都是 HTML 格式的文件，内容是以 ASCII 字符流来传输



浏览器推动的 HTTP/1.0

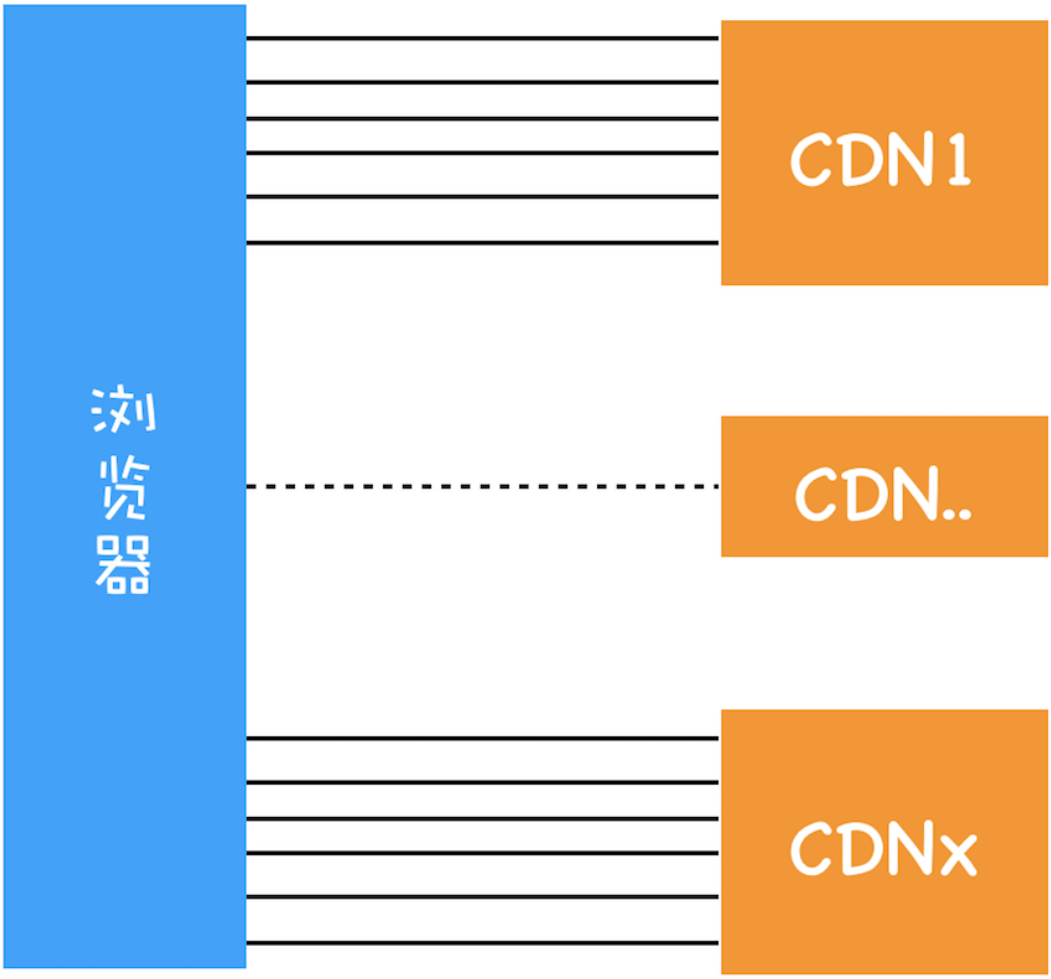
- 请求头和响应头：支持文件类型、压缩、语言版本、编码类型
- 引入了状态码
- 提供了 Cache 机制
- 加入了用户代理字段

HTTP/1.1

- 增加了持久连接：一个 TCP 连接上可以传输多个 HTTP 请求。
- 不成熟的 HTTP 管线化：TCP队头阻塞
- 提供虚拟主机的支持：增加 Host 字段表示当前的域名地址
- 动态生成的内容提供了完美支持：Chunk transfer 机制（Bigpipe）
- Cookie、安全机制

HTTP/1.1 为网络效率做了大量的优化，最核心的有如下三种方式：

1. 增加了持久连接；
2. 浏览器为每个域名最多同时维护 6 个 TCP 持久连接；
3. 使用 CDN 的实现域名分片机制。

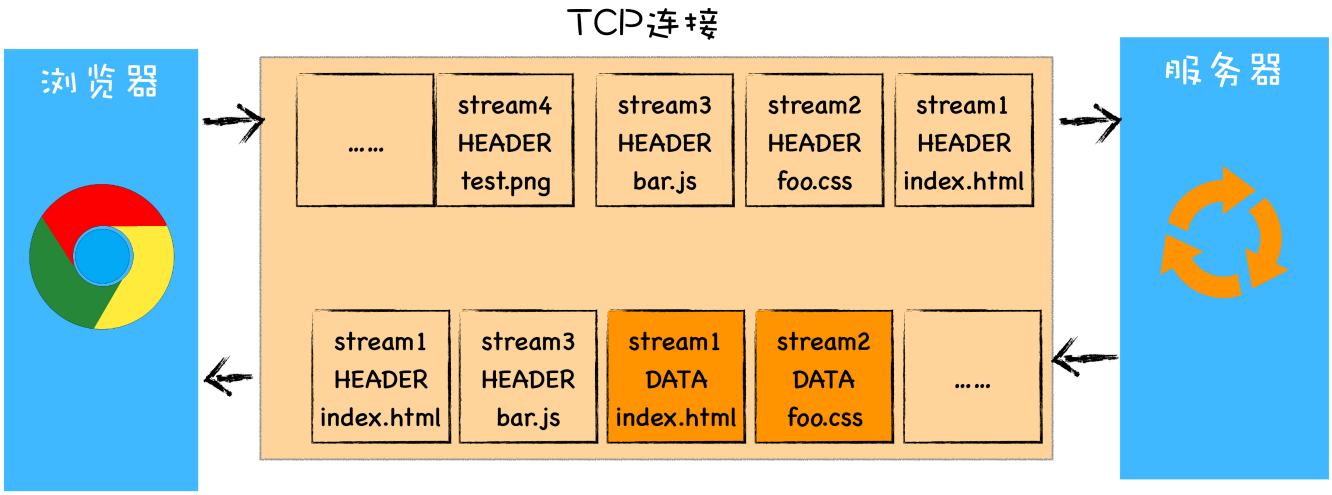


HTTP/1.1 的主要问题：

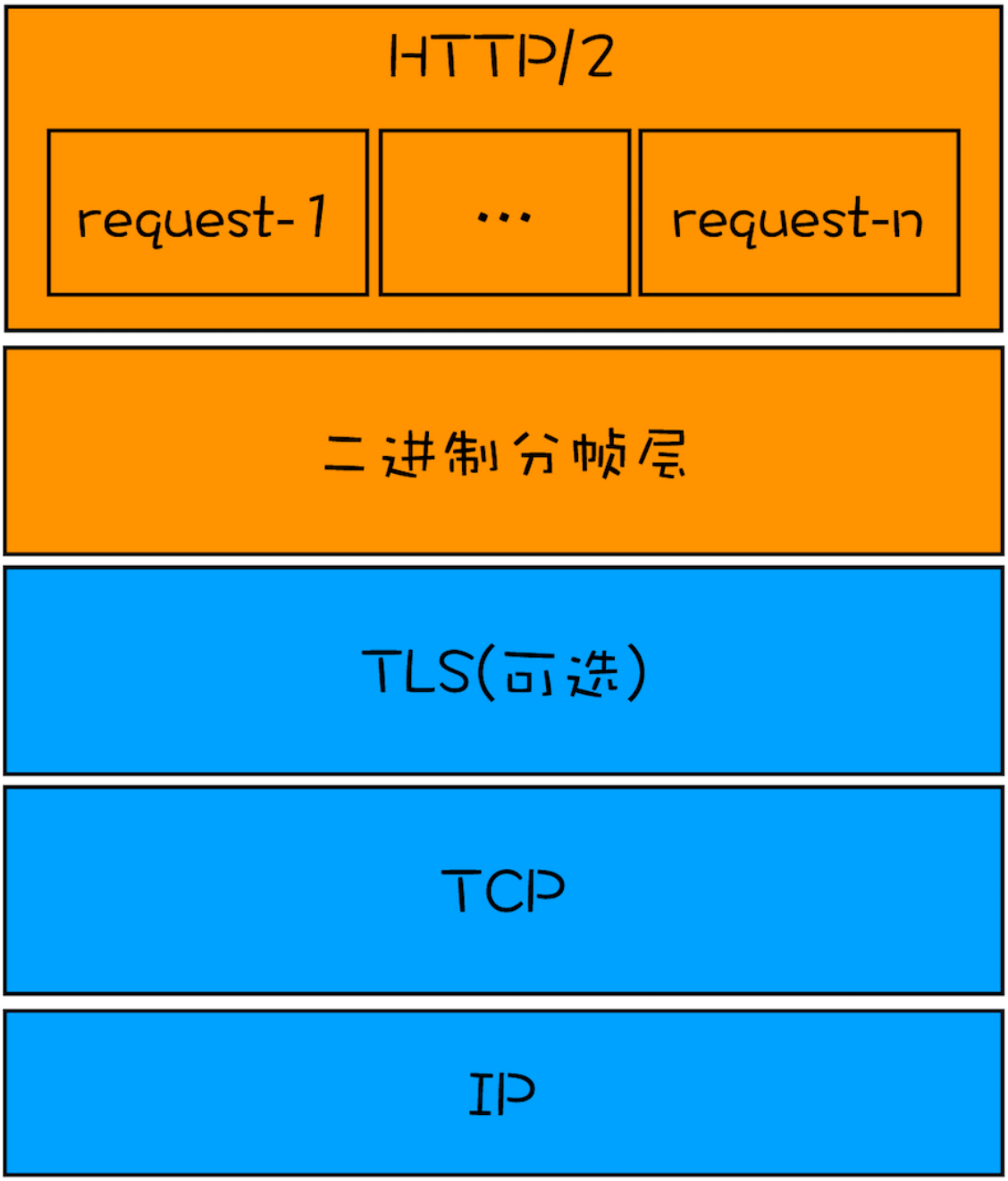
- HTTP/1.1对带宽的利用率却并不理想
- TCP 的慢启动
- 同时开启了多条 TCP 连接，那么这些连接会竞争固定的带宽
- HTTP/1.1 队头阻塞的问题

HTTP/2.0

一个域名只使用一个 TCP 长连接来消除队头阻塞问题


TCP连接
浏览器
服务器
……
stream4
HEADER
test.png
stream3
HEADER
bar.js
stream2
HEADER
foo.css
stream1
HEADER
index.html
stream1
HEADER
index.html
stream3
HEADER
bar.js
stream1
DATA
index.html
stream2
DATA
foo.css
……

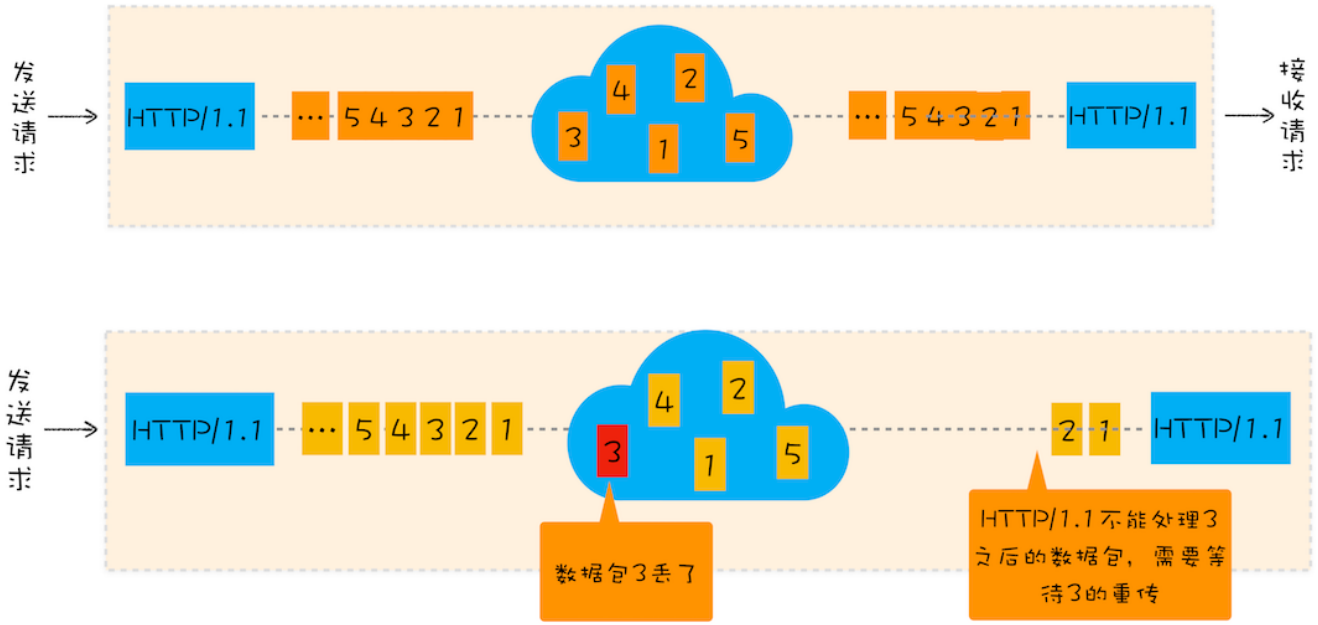
- 多路复用机制



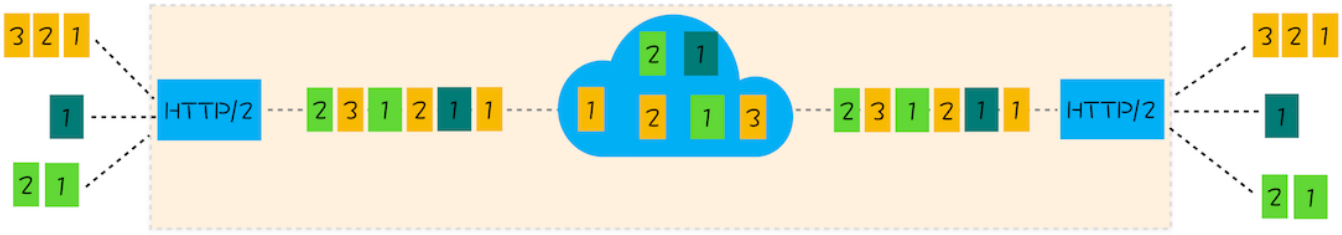
- 可以设置请求的优先级
- 服务器推送
- 头部压缩

HTTP/3.0

- 队头阻塞问题：虽然 HTTP/2 解决了应用层面的队头阻塞问题，不过和 HTTP/1.1 一样，HTTP/2 依然是基于 TCP 协议的，而 TCP 最初就是为了单连接而设计的。你可以把 TCP 连接看成是两台计算机之前的一个虚拟管道，计算机的一端将要传输的数据按照顺序放入管道，最终数据会以相同的顺序出现在管道的另外一头。



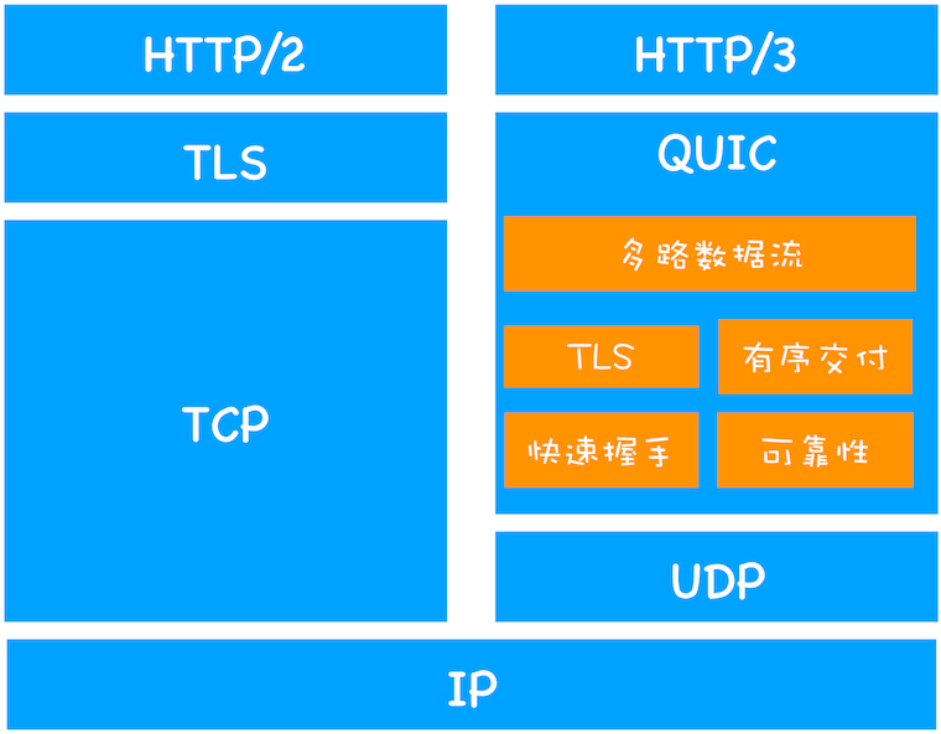
TCP 传输过程中，由于单个数据包的丢失而造成的阻塞称为 TCP 上的队头阻塞。



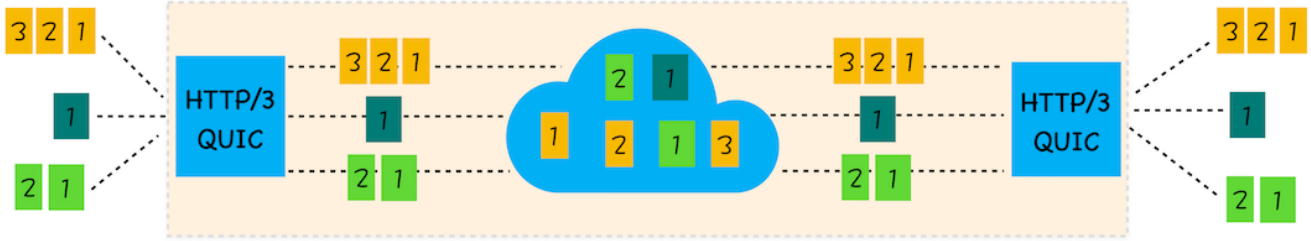
当系统达到了 2% 的丢包率时，HTTP/1.1 的传输效率反而比 HTTP/2 表现得更好。

- TCP 建立连接的延时：需要花掉 3~4 个 RTT（Round Trip Time）
- TCP 协议僵化

基于UDP 的QUIC 协议



- 实现了类似 TCP 的流量控制、传输可靠性的功能
- 集成了 TLS 加密功能
- 实现了快速握手功能
- 实现了 HTTP/2 中的多路复用功能

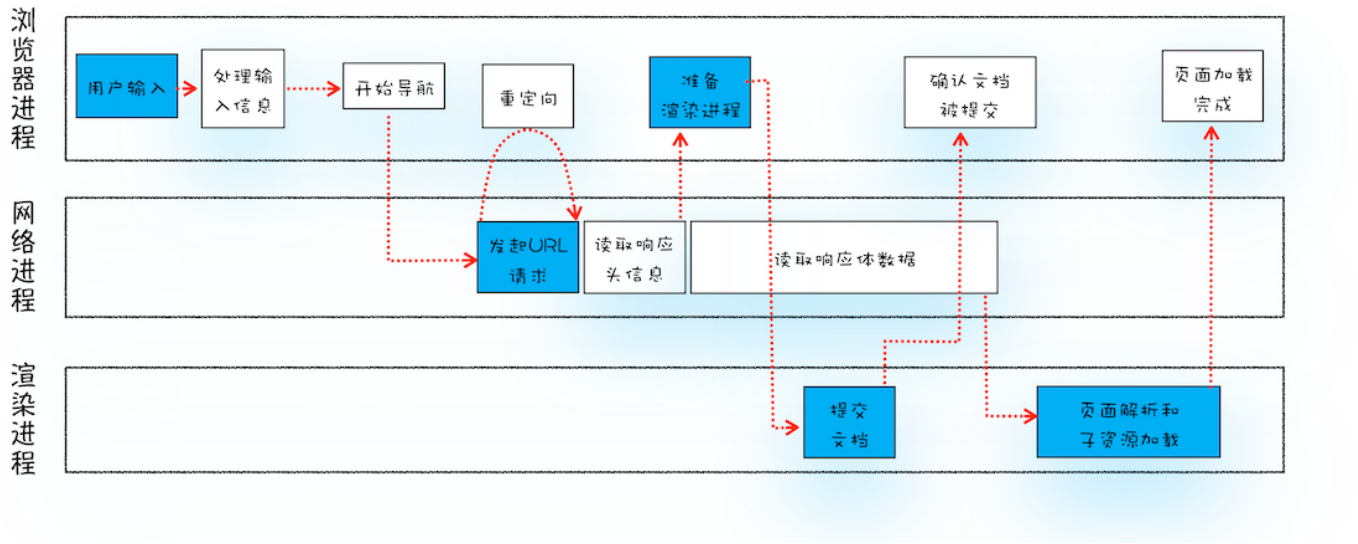


HTTP/3 是个完美的协议，从目前的情况来看，服务器和浏览器端都没有对 HTTP/3 提供比较完整的支持。系统内核对 UDP 的优化远远没有达到 TCP 的优化程度，部署 HTTP/3 也存在着非常大的问题，中间设备僵化的问题。

流程

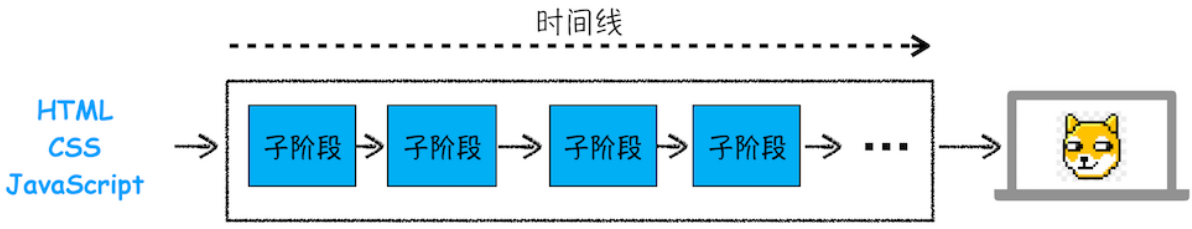
在浏览器里，从输入 URL 到页面展示，这中间发生了什么？

导航



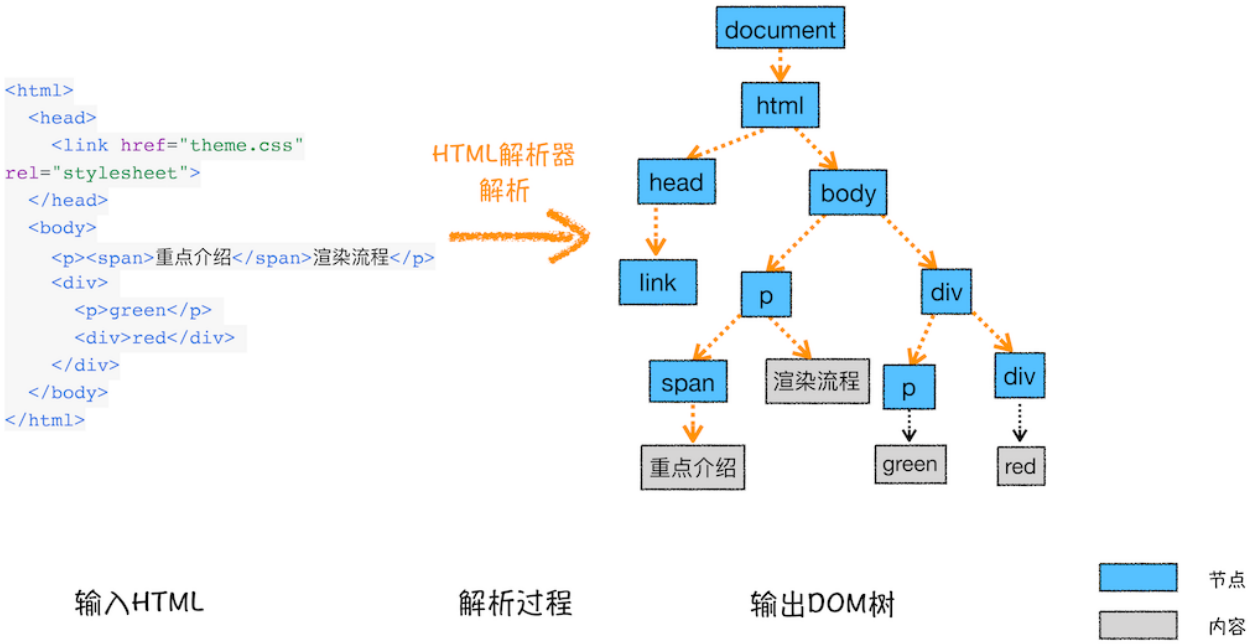
- 首先，浏览器进程接收到用户输入的 URL 请求，浏览器进程便将该 URL 转发给网络进程。
- 然后，在网络进程中发起真正的 URL 请求。接着网络进程接收到了响应头数据，便解析响应头数据，并将数据转发给浏览器进程。
- 浏览器进程接收到网络进程的响应头数据之后，发送“提交导航 (CommitNavigation)”消息到渲染进程；
- 渲染进程接收到“提交导航”的消息之后，便开始准备接收 HTML 数据，接收数据的方式是直接和网络进程建立数据管道；
- 最后渲染进程会向浏览器进程“确认提交”，这是告诉浏览器进程：“已经准备好接受和解析页面数据了”。浏览器进程接收到渲染进程“提交文档”的消息之后，便开始移除之前旧的文档，然后更新浏览器进程中的页面状态。

渲染

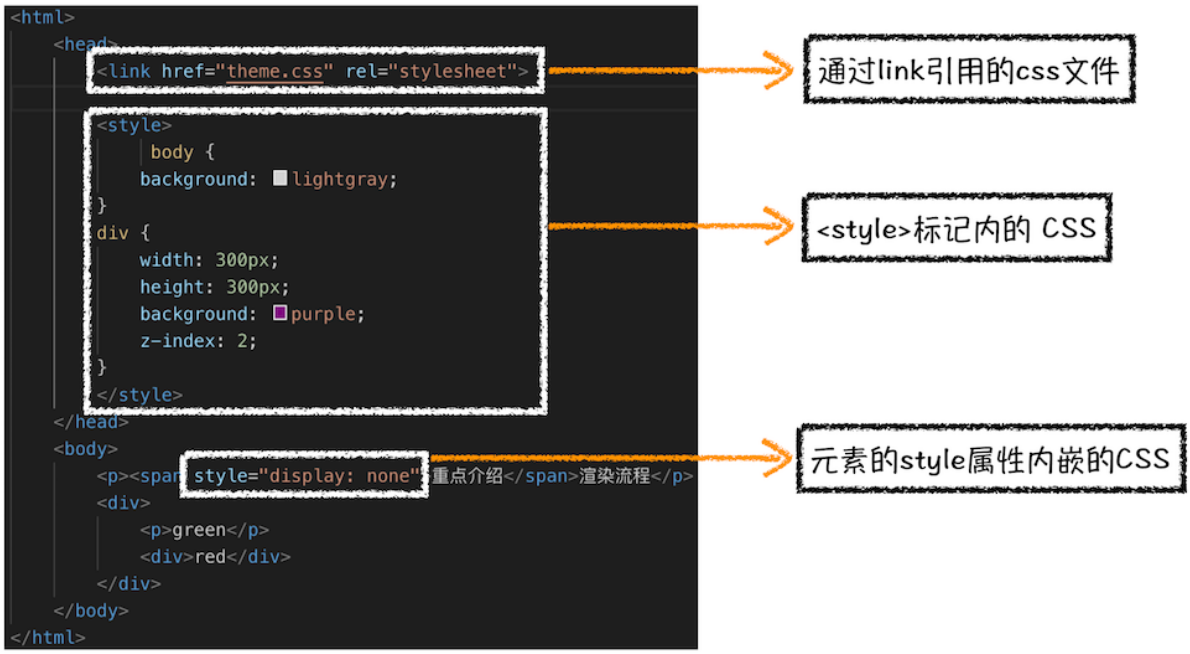


构建 DOM 树

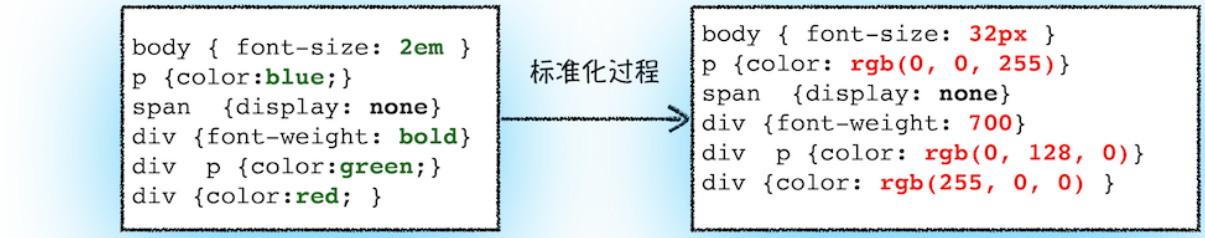
浏览器无法直接理解和使用 HTML，所以需要将 HTML 转换为浏览器能够理解的结构——DOM 树



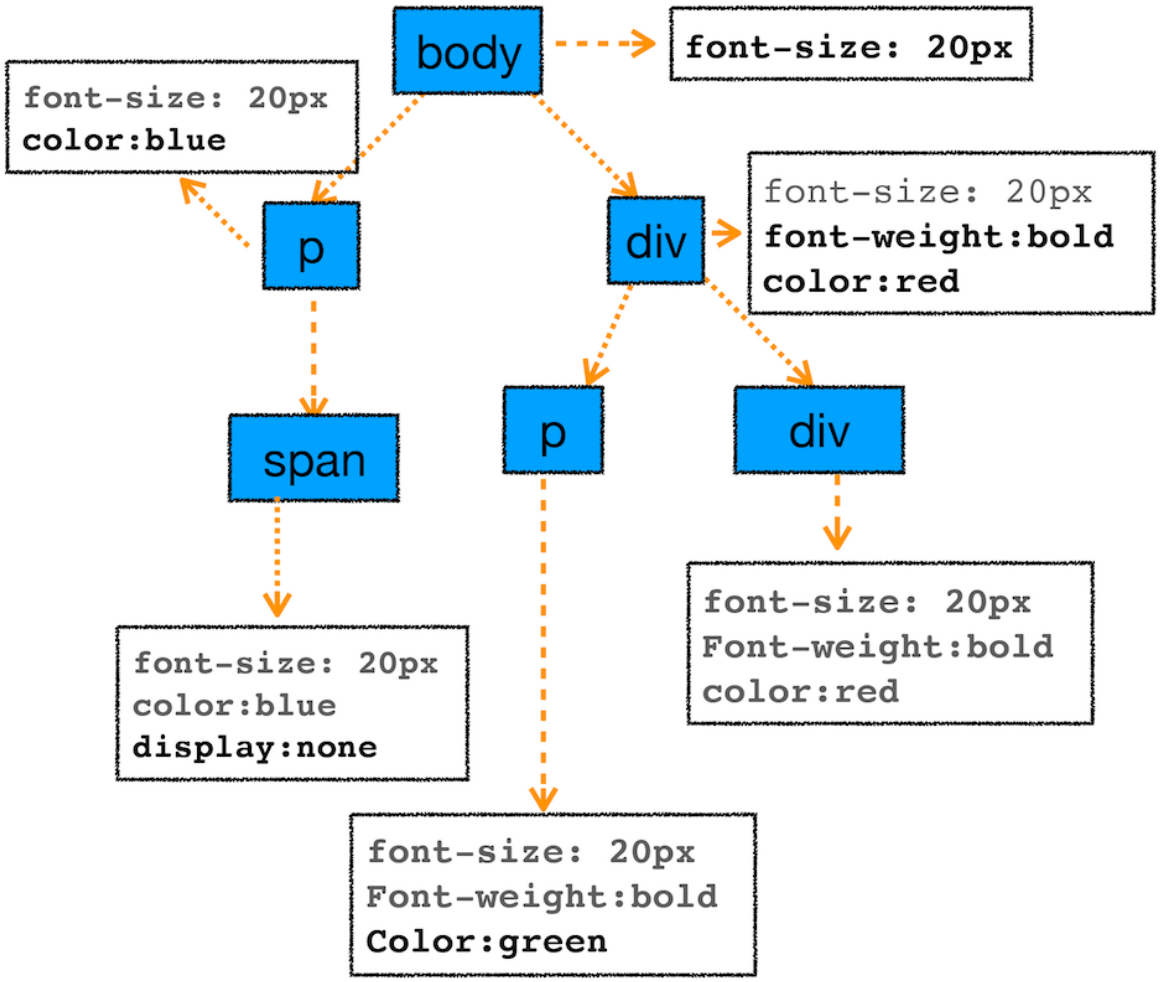
样式计算(继承、层叠)



- 当渲染引擎接收到 CSS 文本时，会执行一个转换操作，将 CSS 文本转换为浏览器可以理解的结构——styleSheets（document.styleSheets）
- 转换样式表中的属性值，使其标准化

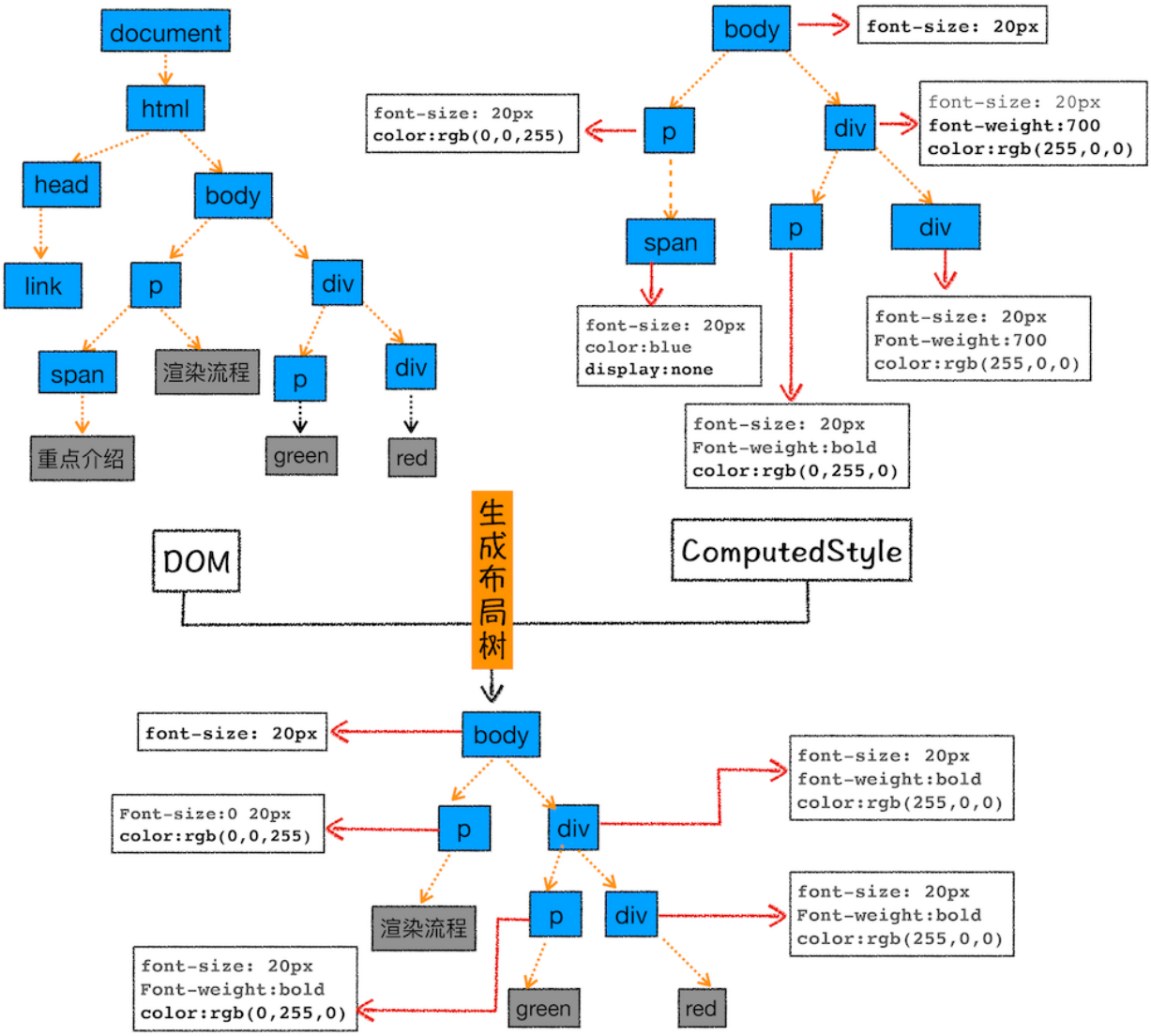


- 计算出 DOM 树中每个节点的具体样式



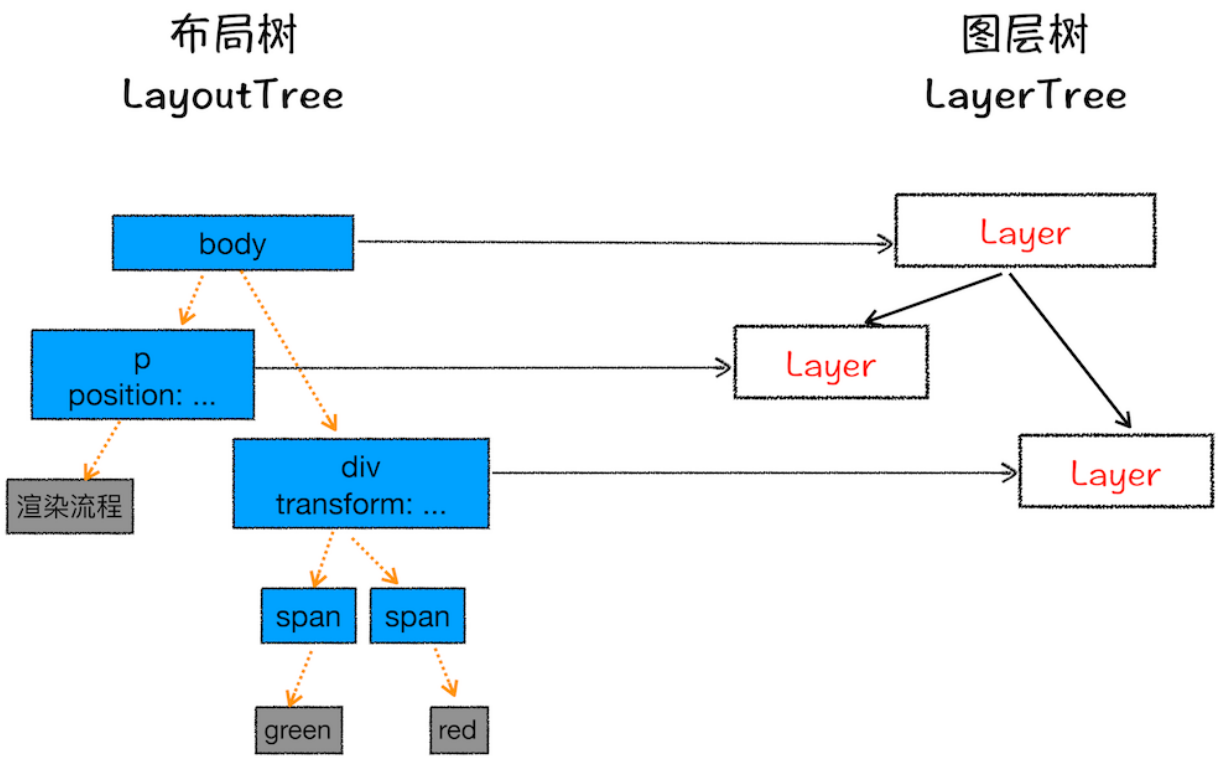
布局阶段

我们有 DOM 树和 DOM 树中元素的样式，但这还不足以显示页面，因为我们还不知道 DOM 元素的几何位置信息。那么接下来就需要计算出 DOM 树中可见元素的几何位置。

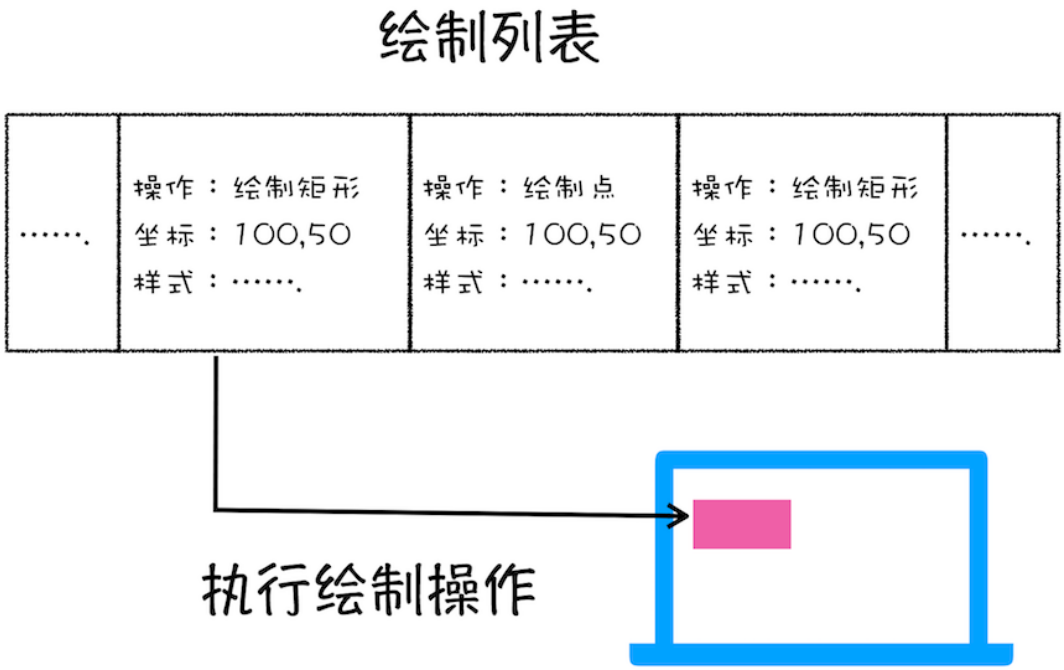


分层、绘制

渲染引擎需要为特定的节点生成专用的图层，并生成一棵对应的图层树（LayerTree）



不是布局树的每个节点都包含一个图层，如果一个节点没有对应的层，那么这个节点就从属于父节点的图层。



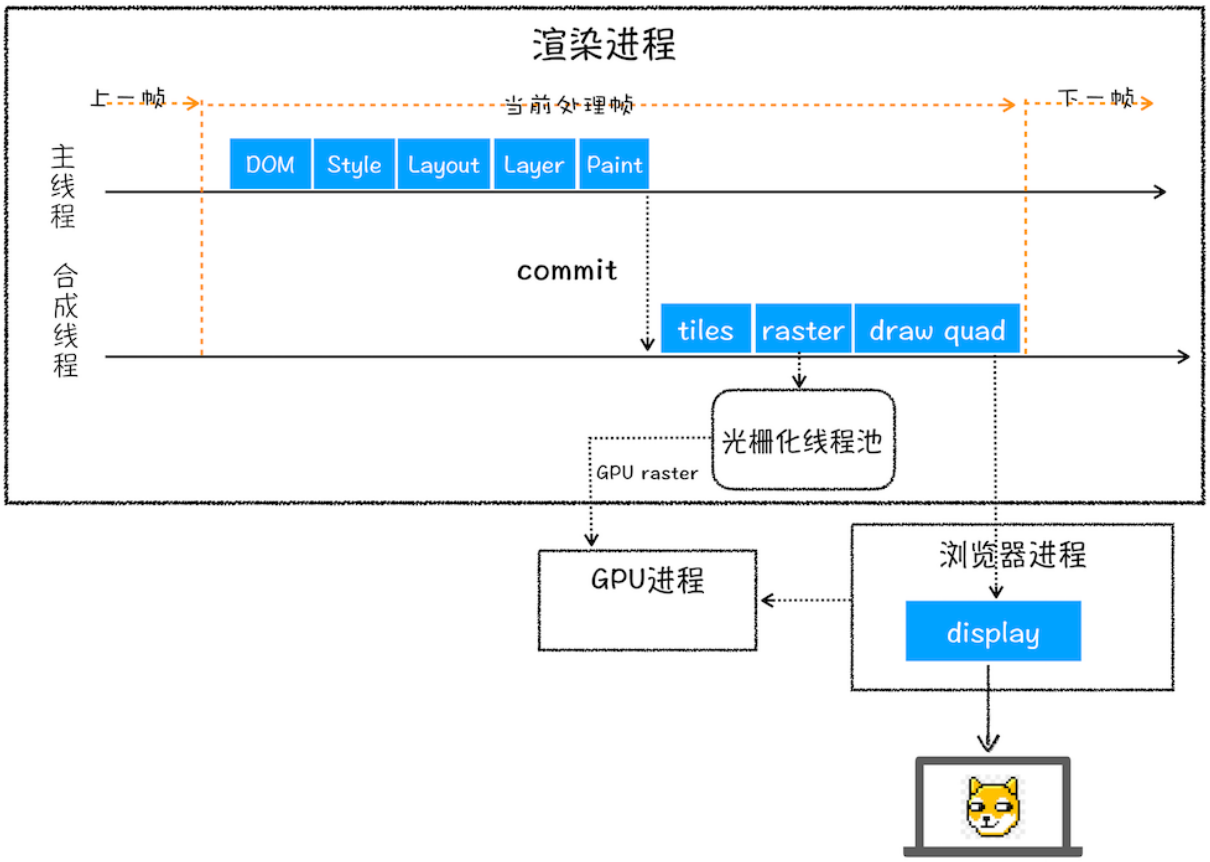
分块、光栅化

绘制列表只是用来记录绘制顺序和绘制指令的列表，而实际上绘制操作是由渲染引擎中的合成线程来完成的。合成线程会将图层划分为图块（tile），合成线程会按照视口附近的图块来优先生成位图，实

际生成位图的操作是由栅格化来执行的。所谓栅格化，是指将图块转换为位图。

合成和显示

一旦所有图块都被光栅化，合成线程就会生成一个绘制图块的命令——“DrawQuad”，然后将该命令提交给浏览器进程。浏览器进程里面有一个叫 viz 的组件，用来接收合成线程发过来的 DrawQuad 命令，然后根据 DrawQuad 命令，将其页面内容绘制到内存中，最后再将内存显示在屏幕上。



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。
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，并生成分层树。
5. 为每个图层生成绘制列表，并将其提交到合成线程。
6. 合成线程将图层分成图块，并在光栅化线程池中将图块转换成位图。
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 消息生成页面，并显示到显示器上。