Web 前端性能

一、网页生成的过程（5 步）

1) HTML 代码转化成 DOM

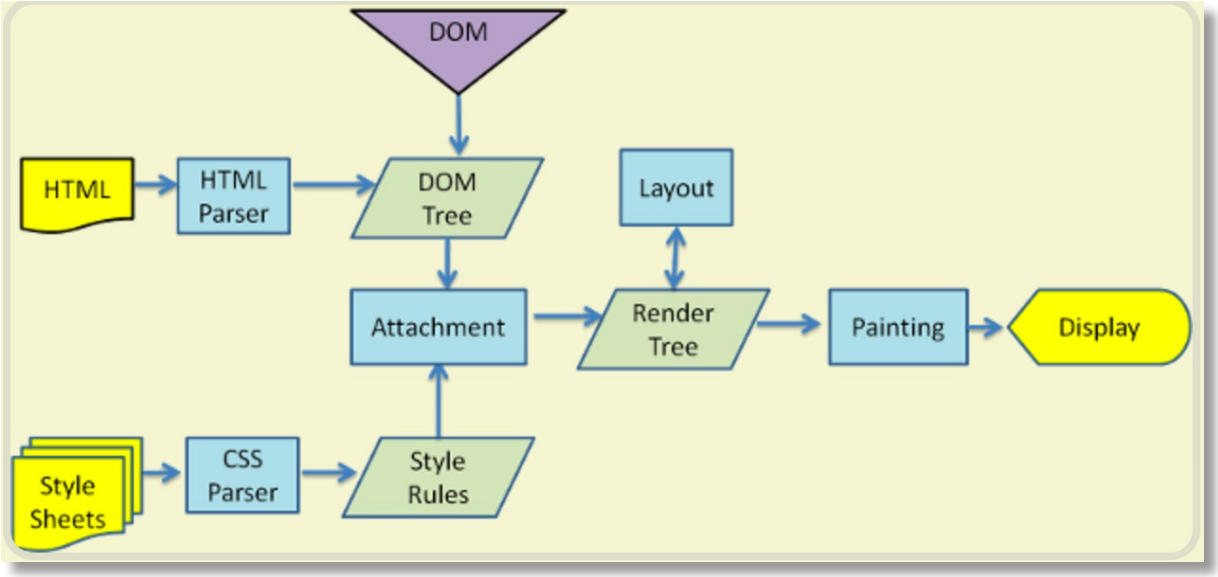
2) CSS 代码转化成 CSSOM（CSS Object Model）

3) 结合 DOM 和 CSSOM，生成一棵渲染树（包含每个节点的视觉信息）

4) 生成布局（layout），即将所有渲染树的所有节点进行平面合成

5) 将布局绘制（paint）在屏幕上

"生成布局"（flow）和"绘制"（paint）这两步，合称为"渲染"（render）。



二、重排和重绘

网页生成的时候，至少会渲染一次。用户访问的过程中，还会不断重新渲染。

以下三种情况，会导致网页重新渲染。

- 修改 DOM
- 修改样式表
- 用户事件（比如鼠标悬停、页面滚动、输入框键入文字、改变窗口大小等等）

重新渲染，就需要重新生成布局和重新绘制。前者叫做"重排"（reflow），后者叫做"重绘"（repaint）。

需要注意的是，"重绘"不一定需要"重排"，比如改变某个网页元素的颜色，就只会触发"重绘"，不会触发"重排"，因为布局没有改变。但是，"重排"必然导致"重绘"，比如改变一个网页元素的位置，就会同时触发"重排"和"重绘"，因为布局改变了。

三、对于性能的影响

重排和重绘会不断触发，这是不可避免的。但是，它们非常耗费资源，是导致网页性能低下的根本原因。

提高网页性能，就是要降低"重排"和"重绘"的频率和成本，尽量少触发重新渲染。

前面提到，DOM 变动和样式变动，都会触发重新渲染。但是，浏览器已经很智能了，会尽量把所有的变动集中在一起，排成一个队列，然后一次性执行，尽量避免多次重新渲染。

```
div.style.color = 'blue';
div.style.marginTop = '30px';
```

上面代码中，div 元素有两个样式变动，但是浏览器只会触发一次重排和重绘。

如果写得不好，就会触发两次重排和重绘。

```
div.style.color = 'blue';
var margin = parseInt(div.style.marginTop);
div.style.marginTop = (margin + 10) + 'px';
```

上面代码对 div 元素设置背景色以后，第二行要求浏览器给出该元素的位置，所以浏览器不得不立即重排。

一般来说，样式的写操作之后，如果有下面这些属性的读操作，都会引发浏览器立即重新渲染。

- offsetTop/offsetLeft/offsetWidth/offsetHeight
- scrollTop/scrollLeft/scrollWidth/scrollHeight
- clientTop/clientLeft/clientWidth/clientHeight
- getComputedStyle()

所以，从性能角度考虑，尽量不要把读操作和写操作，放在一个语句里面。

```
// bad
div.style.left = div.offsetLeft + 10 + "px";
div.style.top = div.offsetTop + 10 + "px";

// good
var left = div.offsetLeft;
var top  = div.offsetTop;
div.style.left = left + 10 + "px";
div.style.top = top + 10 + "px";
```

一般的规则是：

- 样式表越简单，重排和重绘就越快。
- 重排和重绘的 DOM 元素层级越高，成本就越高。
- table 元素的重排和重绘成本，要高于 div 元素

四、提高性能

1. 减少 HTTP 请求

而当我们请求的网页文件中有很多图片、CSS、JS 甚至音乐等信息时，将会频繁的与服务器建立连接，与释放连接，这必定会造成资

源的浪费，且每个 HTTP 请求都会对服务器和浏览器产生性能负担。

网速相同的条件下，下载一个 100KB 的图片比下载两个 50KB 的图片要快。

解决办法：

合并 CSS、合并 JavaScript、合并图片。将浏览器一次访问需要的 javascript 和 CSS 合并成一个文件，这样浏览器就只需要一次请求。图片也可以合并，多张图片合并成一张，如果每张图片都有不同的超链接，可通过 CSS 偏移响应鼠标点击操作，构造不同的 URL。

2. 减少 DNS 查找次数(DNS 缓存)

3. 使用内容分发网络(CDN – Content delivery network)

内容分发网络是由一系列分散到各个不同地理位置上的 Web 服务器组成的，它提高了网站内容的传输速度。用于向用户传输内容的服务器主要是根据和用户在网络上的靠近程度来指定的。使用 CDN，就相当于在离你最近的地方，放置一台性能好、连接顺畅的副本服务器，让你能够以最近的距离，最快的速度获取内容。有个缺点是：CDN 服务成本却非常高，需要建立多台服务器。

4. 添加 Expire/Cache-Control 头

通过使用 Expires 或者 Cache-Control 就可以使请求的内容具有缓存性，它避免了接下来的页面访问中不必要的 HTTP 请求。Expires 头的内容是一个时间值，值就是资源在本地的过期时间。在当前时间还没有超过缓存资源的过期时间时，就直接使用这一缓存的资源，不会发送 HTTP 请求。Cache-Control 的作用也是类似的，只不过它的值是一个表示距离缓存过期的一个秒数时间。

5. 启用 Gzip 压缩

通过减小 HTTP 响应的大小也可以节省 HTTP 响应时间。Gzip 可以压缩所有可能的文件类型，是减少文件体积、增加用户体验的简单方法。

从 HTTP/1.1 开始，web 客户端都默认支持 HTTP 请求中有 Accept-Encoding 文件头的压缩格式：Accept-Encoding: gzip

如果 web 服务器在请求的文件头中检测到上面的代码，就会以客户端列出的方式压缩响应内容。Web 服务器把压缩方式通过响应文件头中的 Content-Encoding 来返回给浏览器。

Content-Encoding: gzip

6. 将 CSS 和 JS 放到外部文件中引用，CSS 放头，JS 放尾

浏览器会在下载完成全部 CSS 之后才对整个页面进行渲染，因此最好的做法是将 CSS 放在页面最上面，让浏览器尽快下载 CSS。如果将 CSS 放在其他地方比如 BODY 中，则浏览器有可能还未下载和解析到 CSS 就已经开始渲染页面了，这就导致页面由无 CSS 状态跳转到 CSS 状态，用户体验比较糟糕，所以可以考虑将 CSS 放在 HEAD 中。

Javascript 则相反，浏览器在加载 javascript 后立即执行，有可能会阻塞整个页面，造成页面显示缓慢，因此 javascript 最好放在页面最下面。但如果页面解析时就需要用到 javascript，这时放到底部就不合适了。

7. 压缩 JavaScript 和 CSS

压缩是指从去除代码不必要的字符减少文件大小从而节省下载时间。

方法：

1.去除不必要的空白符（空格、换行、tab 缩进）、格式符、注释符

2.简写方法名、参数名来压缩 js 脚本

在 JavaScript 中，由于需要下载的文件体积变小了从而节省了响应时间。