你对前端性能优化有什么了解?

百度二面面试题。在你面试生涯中，这道题可能会迟到，但永远不会缺席。性能优化是个永恒的话题，风里雨里，在面试官那等你~不要怂、一起上

性能优化涉及方方面面，主要体现在如下几点（已竭尽所能的整理了这么多，若仍然有整理不到的地方，望大家指出，笔芯）

---

1. JS相关

- 首先你需要了解js时间线，看完你就知道了为什么js要放到最下面加载（默认了解，不了解的百度“js时间线”，10条）
- 懒执行。懒执行就是将某些逻辑延迟到使用时再计算。该技术可以用于首屏优化，对于某些耗时逻辑并不需要在首屏就使用的，就可以使用懒执行。懒执行需要唤醒，一般可以通过定时器或者事件的调用来唤醒。
- 对DOM动画使用CSS类而不是JavaScript。
- 使用快速DOM遍历document.getElementById()。
- 指针引用存储到浏览器内对象中
- 批量更改DOM，尤其是在更新样式时
- 在将DOM添加到页面之前单独构建DOM。
- 在可滚动的DIV中使用缓冲的DOM。

2. 渲染相关

- **渲染过程优化**

> 首先了解一下渲染机制：
>
> 1） 处理 HTML 并构建 DOM 树。
>
> 2） 处理 CSS 构建 CSSOM 树。
>
> 3） 将 DOM 与 CSSOM 合并成一个渲染树。
>
> 4） 根据渲染树来布局，计算每个节点的位置。
>
> 5） 调用 GPU 绘制，合成图层，显示在屏幕上。

在构建 CSSOM 树时，会阻塞渲染，直至 CSSOM 树构建完成。并且构建 CSSOM 树是一个十分消耗性能的过程，所以应该尽量保证层级扁平，减少过度层叠，越是具体的 CSS 选择器，执行速度越慢。

- **重排和重绘（回流）**

  重绘：和回流是渲染步骤中的一小节，但是这两个步骤对于性能影响很大。

  重绘是当节点需要更改外观而不会影响布局的，比如改变 color 就叫称为重绘

  回流：是布局或者几何属性需要改变就称为回流。

回流必定会发生重绘，重绘不一定会引发回流。回流所需的成本比重绘高的多，改变深层次的节点很可能导致父节点的一系列回流。

下列情况会发生重排：

1） 页面初始渲染

2） 添加/删除可见DOM元素

3） 改变元素位置

4） 改变元素尺寸（宽、高、内外边距、边框等）

5） 改变元素内容（文本或图片等）

6） 改变窗口尺寸

不同的条件下发生重排的范围及程度会不同 某些情况甚至会重排整个页面，比如resize 或者 scroll 。

减少重绘和重绘：

1） 使用 translate 替代 top

2） 使用 visibility 替换 display: none ，因为前者只会引起重绘，后者会引发回流（改变了布局）

3） 把 DOM 离线后修改，比如：先把 DOM 给 display:none (有一次 Reflow)，然后你修改100次，然后再把它显示出来

4） 不要把 DOM 结点的属性值放在一个循环里当成循环里的变量

5） 不要使用 table 布局，可能很小的一个小改动会造成整个 table 的重新布局

6） 动画实现的速度的选择，动画速度越快，回流次数越多，也可以选择使用 requestAnimationFrame

7） CSS 选择符从右往左匹配查找，避免 DOM 深度过深

将频繁运行的动画变为图层，图层能够阻止该节点回流影响别的元素。比如对于 video 标签，浏览器会自动将该节点变为图层。

- **图层**

一般来说，可以把普通文档流看成一个图层。特定的属性可以生成一个新的图层。不同的图层渲染互不影响，所以对于某些频繁需要渲染的建议单独生成一个新图层，提高性能。但也不能生成过多的图层，会引起反作用。

通过以下几个常用属性可以生成新图层：

1) 3D 变换：translate3d、translateZ

2) will-change

3) video、iframe 标签

4) 通过动画实现的 opacity 动画转换

5) position: fixed

- **懒加载**

  懒加载就是将不关键的资源延后加载懒加载的原理就是只加载自定义区域（通常是可视区域，但也可以是即将进入可视区域）内需要加载的东西。对于图片来说，先设置图片标签的 src 属性为一张占位图，将真实的图片资源放入一个自定义属性中，当进入自定义区域时，就将自定义属性替换为 src 属性，这样图片就会去下载资源，实现了图片懒加载。

  懒加载不仅可以用于图片，也可以使用在别的资源上。比如进入可视区域才开始播放视频等等。

3. 文件优化

- **图片优化**

  1) 计算图片大小

  对于一张 100 * 100 像素的图片来说，图像上有 10000 个像素点，如果每个像素的值是 RGBA 存储的话，那么也就是说每个像素有 4 个通道，每个通道 1 个字节（8 位 = 1个字节），所以该图片大小大概为 39KB（10000 * 1 * 4 / 1024）。

  但是在实际项目中，一张图片可能并不需要使用那么多颜色去显示，我们可以通过减少每个像素的调色板来相应缩小图片的大小。

  了解了如何计算图片大小的知识，那么对于如何优化图片，想必大家已经有 2 个思路了：

  a. 减少像素点

  b. 减少每个像素点能够显示的颜色

  2) 图片加载优化

  不用图片。很多时候会使用到很多修饰类图片，其实这类修饰图片完全可以用 CSS 去代替。

  b. 对于移动端来说，屏幕宽度就那么点，完全没有必要去加载原图浪费带宽。
  一般图片都用 CDN 加载，可以计算出适配屏幕的宽度，然后去请求相应裁剪好的图片。

  c. 小图使用 base64 格式

```
        d. 将多个图标文件整合到一张图片中（雪碧图）

        e. 选择正确的图片格式：
            对于能够显示 WebP 格式的浏览器尽量使用 WebP 格式。
            因为 WebP 格式具有更好的图像数据压缩算法，能带来更小的图片体积,而且拥有肉眼识别无差异的图像质量，缺点就是兼容性并不好；
            小图使用 PNG，其实对于大部分图标这类图片，完全可以使用 SVG 代替；
            照片使用 JPEG；
```

- **其他文件优化**

```
    1)  CSS 文件放在 head 中

    2)  服务端开启文件压缩功能

    3)  将 script 标签放在 body 底部，因为 JS 文件执行会阻塞渲染。当然也可以把 script 标签放在任意位置然后加上 defer ，表示该文件会并行下载，但是会放到 HTML 解析完成后顺序执行。对于没有任何依赖的 JS 文件可以加上 async ，表示加载和渲染后续文档元素的过程将和 JS 文件的加载与执行并行无序进行。

    4)  执行 JS 代码过长会卡住渲染，对于需要很多时间计算的代码可以考虑使用 Webworker。Webworker可以让我们另开一个线程执行脚本而不影响渲染。
```

- **CDN**

  静态资源尽量使用 CDN 加载，由于浏览器对于单个域名有并发请求上限，可以考虑使用多个 CDN 域名。对于 CDN 加载静态资源需要注意 CDN 域名要与主站不同，否则每次请求都会带上主站的 Cookie。

4. 网络相关

- **缓存**

  缓存对于前端性能优化来说是个很重要的点，良好的缓存策略可以降低资源的重复加载提高网页的整体加载速度。

```
1)  实现强缓存可以通过两种响应头实现：Expires 和 Cache-Control

2)  如果缓存过期了，我们就可以使用Last-Modified和If-Modified-Since、ETag和If-None-Match来解决问题。如果缓存有效会返回 304。同样也需要客户端和服务器共同实现。

3)  择合适的缓存策略
```

对于大部分的场景都可以使用强缓存配合协商缓存解决，但是在一些特殊的地方可能需要选择特殊的缓存策略

a. 对于某些不需要缓存的资源，可以使用 Cache-control: no-store ，表示该资源不需要缓存

b. 对于频繁变动的资源，可以使用 Cache-Control: no-cache 并配合 ETag 使用，表示该资源已被缓存，但是每次都会发送请求询问资源是否更新。

c. 对于代码文件来说，通常使用 Cache-Control: max-age=31536000 并配合策略缓存使用，然后对文件进行指纹处理，一旦文件名变动就会立刻下载新的文件。

- **使用HTTP/2.0**

  因为浏览器会有并发请求限制，在 HTTP / 1.1 时代，每个请求都需要建立和断开，消耗了好几个 RTT 时间，并且由于 TCP 慢启动的原因，加载体积大的文件会需要更多的时间。

  在 HTTP / 2.0 中引入了多路复用，能够让多个请求使用同一个 TCP 链接，极大的加快了网页的加载速度。并且还支持 Header 压缩，进一步的减少了请求的数据大小。

- **预加载**

  开发中，可能会遇到这样的情况。有些资源不需要马上用到，但是希望尽早获取，这时候就可以使用预加载。

```
<link rel="preload" href="http://www.aaa.com">
```

预加载其实是声明式的 fetch ，强制浏览器请求资源，并且不会阻塞 onload 事件，可以使用以下代码开启预加载。

- **预渲染**

  以通过预渲染将下载的文件预先在后台渲染，可以使用以下代码开启预渲染

```
<link rel="prerender" href="http://www.aaa.com">
```

预渲染虽然可以提高页面的加载速度，但是要确保该页面百分百会被用户在之后打开，否则就白白浪费资源去渲染。

5. 构建工具

用webpack优化项目

- 对于 Webpack4，打包项目使用 production 模式，这样会自动开启代码压缩
- 使用 ES6 模块来开启 tree shaking，这个技术可以移除没有使用的代码
- 优化图片，对于小图可以使用 base64 的方式写入文件中
- 按照路由拆分代码，实现按需加载
- 给打包出来的文件名添加哈希，实现浏览器缓存文件