前端面试必备八股文——性能优化（6题）

图片懒加载原理

图片懒加载也叫延迟加载，只加载当前屏幕的图片，可视区域外的图片不会进行加载，只有当屏幕滚动的时候才加载。

特点：

- 提高网页加载速度
- 减少后台服务器压力
- 提升用户体验

原理

- 将图片地址存储到 `data-xxx` 属性上
- 判断图片是否在可视区域
- 如果在，就设置图片 `src`
- 绑定 `scroll` 监听事件

节流和防抖

节流

节流是一种常用的性能优化技术，它可以限制函数的执行频率，避免过多的重复操作，提升页面的响应速度。

函数在 n 秒内只执行一次，如果多次触发，则忽略执行。

应用场景：

- 拖拽场景
- scroll场景
- 窗口大小调整

「手写代码-节流」

防抖

防抖函数可以将多次高频率触发的函数执行合并成一次，并在指定的时间间隔后执行一次。通常在处理输入框、滚动等事件时使用，避免频繁触发事件导致页面卡顿等问题。

函数在 n 秒后再执行，如果 n 秒内被触发，重新计时，保证最后一次触发事件 n 秒后才执行。

应用场景：

- 输入框搜索
- 表单提交按钮
- 文本器保存

「手写代码-防抖」

SPA首屏为什么加载慢?

`SPA` 首屏加载慢可能有以下原因：

- **JavaScript文件过大**：SPA通常有很多 JavaScript 文件，如果这些文件的大小过大或加载速度慢，就会导致首屏加载缓慢。可以通过代码分割和打包、使用CDN等方式来优化加载速度。
- **数据请求过多或数据请求太慢**：SPA通过 AJAX 或 Fetch 等方式从后端获取数据，如果数据请求过多或数据请求太慢，也会导致首屏加载缓慢。可以通过减少数据请求、使用数据缓存、优化数据接口等方式来优化数据请求速度。
- **大量图片加载慢**：如果首屏需要加载大量图片，而这些图片大小过大或加载速度慢，也会导致首屏加载缓慢。可以通过图片压缩、使用图片懒加载等方式来优化图片加载速度。
- **过多的渲染和重绘操作**：如果在首屏加载时进行大量的渲染和重绘操作，也会导致首屏加载缓慢。可以通过尽可能少的DOM操作、使用CSS3动画代替JS动画等方式来优化渲染和重绘操作。
- **网络问题**：网络问题也会影响SPA首屏加载速度，比如网络延迟、丢包等。可以通过使用CDN、使用HTTP/2等方式来优化网络问题。

为什么要做性能优化

性能优化是为了提高网页的加载速度和相应速度，给用户带来更好的体验和用户满意度，同时还能减少服务器的负载压力，以此来提升程序的稳定性，具体有以下几个因素：

- 提高用户体验
- 增加页面访问量
- 提高搜索引擎排名
- 减少服务器压力
- 节约成本
- 提高用户留存率

常见性能优化有哪些关键指标?

- **首屏加载时间First Contentful Paint（FCP）**：首次内容绘制时间，指浏览器首次绘制页面中至少一个文本、图像、非白色背景色的 `canvas/svg` 元素等的时间，代表页面首屏加载的时间点。
- **首次绘制时间First Paint（FP）**：首次绘制时间，指浏览器首次在屏幕上渲染像素的时间，代表页面开始渲染的时间点。
- **最大内容绘制时间Largest Contentful Paint（LCP）**：最大内容绘制时间，指页面上最大的可见元素（文本、图像、视频等）绘制完成的时间，代表用户视觉上感知到页面加载完成的时间点。
- **用户可交互时间Time to Interactive（TTI）**：可交互时间，指页面加载完成并且用户能够与页面进行交互的时间，代表用户可以开始操作页面的时间点。
- **页面总阻塞时间Total Blocking Time (TBT)**：页面上出现阻塞的时间，指在页面变得完全交互之前，用户与页面上的元素交互时出现阻塞的时间。TBT应该尽可能小，通常应该在300毫秒以内。
- **搜索引擎优化Search Engine Optimization (SEO)**：网站在搜索引擎中的排名和可见性。评分范围从0到100，100分表示网站符合所有SEO最佳实践。

除此之外还有常见的 `258` 原则、 `GOOGLE` 团队建议

258原则

- **2**：页面的加载时间应该控制在2秒以内，这是用户能够接受的最短时间。
- **5**：页面的加载时间在5秒以内，用户对页面加载速度的不满意度开始上升。
- **8**：页面的加载时间超过8秒，用户的流失率将急剧增加，用户很可能会放弃访问该页面。

性能优化方式有哪些

HTML&CSS

- 减少 `DOM` 数量，减轻浏览器渲染计算负担。
- 使用异步和延迟加载 `js` 文件，避免 `js` 文件阻塞页面渲染
- 压缩 `HTML、CSS` 代码体积，删除不要的代码，合并 `CSS` 文件，减少 `HTTP` 请求次数和请求大小。
- 减少 `CSS` 选择器的复杂程度，复杂度与阿高浏览器解析时间越长。
- 避免使用 `CSS` 表达式在 `javascript` 代码中
- 使用 `css` 渲染合成层如 `transform` 、 `opacity` 、 `will-change` 等，提高页面相应速度减少卡顿现象。
- 动画使用 `CSS3` 过渡，减少动画复杂度，还可以使用硬件加速。

JS

- 减少 `DOM` 操作数量

- 避免使用 `with` 语句、 `eval` 函数，避免引擎难以优化。
- 尽量使用原生方法，执行效率高。
- 将 `js` 文件放到文件页面底部，避免阻塞页面渲染
- 使用事件委托，减少事件绑定次数。
- 合理使用缓存，避免重复请求数据。

Vue

- 合理使用 `watch` 和 `computed` ，数据变化就会执行，避免使用太多，减少不必要的开销
- 合理使用组件，提高代码可维护性的同事也会降低代码组件的耦合性
- 使用路由懒加载，在需要的时候才会进行加载，避免一次性加载太多路由，导致页面阻塞
- 使用 `Vuex` 缓存数据
- 合理使用 `mixins` ，抽离公共代码封装成模块，避免重复代码。
- 合理使用 `v-if` 、 `v-show`
- `v-for` 不要和 `v-if` 一起使用， `v-for` 的优先级会比 `v-if` 高
- `v-for` 中不要用 `index` 做 `key` ，要保证 `key` 的唯一性
- 使用异步组件，避免一次性加载太多组件
- 避免使用 `v-html` ，存在安全问风险和性能问题，可以使用 `v-text`
- 使用 `keep-alive` 缓存组件，避免组件重复加载

Webpack优化

- 代码切割，使用 `code splitting` 将代码进行分割，避免将所有代码打包到一个文件，减少响应体积。
- 按需加载代码，在使用使用的时候加载代码。
- 压缩代码体积，可以减小代码体积
- 优化静态资源，使用字体图标、雪碧图、webp格式的图片、svg图标等
- 使用 `Tree Shaking` 删除未被引用的代码
- 开启 `gzip` 压缩
- 静态资源使用 `CDN` 加载，减少服务器压力

网络优化

- 使用 `HTTP/2`
- 减少、合并 `HTTP` 请求，通过合并 `CSS、JS` 文件、精灵图等方式减少请求数量。

- 压缩文件， 开启 `nginx` ， `Gzip` 对静态资源压缩
- 使用 `HTTP` 缓存，如强缓存、协商缓存
- 使用 `CDN` ，将网站资源分布到各地服务器上，减少访问延迟