前端性能优化7种手段

1.减少http请求数量

- 合理的合并：1.合并js，css文件
- 图片处理
  - 雪碧图
  - base64
  - 使用字体图标icon
- 减少重定向，除非必要请使用301永久重定向
- 使用缓存
  - 强缓存
  - 协商缓存
- 不适用CSS的@import
- 避免使用空的src和href
  - a标签设置空的href，会重定向到当前的页面地址
  - form设置空的method，会提交表单到当前的页面地址
  - img标签的src属性
- 使用localstorage/indexedDB：本地storage缓存资源，减少http请求

2.减小资源大小

- 对资源进行压缩
  - HTML压缩：压缩在文本文件中有意义，但是不在界面显示的字符，包括空格，制表符，换行符
  - CSS压缩：包括无效代码删除和CSS语义合并
  - JS压缩与混乱
    - 无效字符及注释的删除
    - 代码语义的缩减和优化，降低代码可读性，实现代码保护
  - 图片压缩：针对真实图片的情况，舍弃一些相对无关紧要的色彩信息
- webp：安卓下可使用webp格式的图片，更优的图像数据压缩算法，带来更小的图片体积，同质量下会比jpg、png小25%
- 开启gizp：http协议上的gzip编码是一种有效的压缩技术，可以将网页中的内容进行压缩，同时在数据传输中也可以压缩文本内容

3.优化网络连接

- 使用CDN：根据当前ip地址，url内容，ip节点服务器的状况将请求重新导向到离用户最近的可负载的服务强上，得到更好的用户体验
- 使用DNS预解析：dns-prefetch，在浏览器空闲时间内的动作，不会影响页面的性能
- 并发请求/并行连接：常说的域名发散，通过将多个域名指向同一个ip地址。浏览器下的同一域名下的并发数最大是6
- 持久连接：开启connection: keep-alive，1.1是默认开启的
- 升级http2：多路复用，服务器推送，压缩头部字段
- 合理使用cookie：减少cookie的数量，降低网络资源消耗

4.优化资源加载

- 资源加载位置
  - CSS文件放在head中，先外链，后本页
  - JS文件加载body闭合标签之前，使用外链
  - body中不要写style标签和script标签
- 资源加载时机
  - 异步scipt标签
    - defer
    - async
  - 模块按需加载：在单页面应用(SPA)中路由懒加载，减少首页加载的dom数量，减轻首页加载资源的压力(import()异步动态加载，webpack的require.ensure)
  - 使用资源预加载preload和预读取prefetch
    - preload: 让浏览器提前加载指定资源，需要执行是再执行，可以加速本页面的加载速度，一定会被用到
    - prefetch: 加载可能会使用到的资源，可能不会被使用
  - 资源懒加载和资源预加载
    - 资源延迟加载/资源懒加载: 延迟加载资源或在符合某些条件时才加载某些资源，如图片懒加载技术(可使用clientHeight，getBoundingClientRect()，IntersectionObserver等方法)
    - 资源预加载: 提前加载用户所需的资源，保证可以直接使用
  - 一种错峰操作，在浏览器空闲时候加载资源优化网络性能

5.减少重绘回流

- 样式设置
  - 避免使用层级较深或复杂的选择器，方便计算与查找，提升渲染效率
  - 避免使用CSS表达式，在页面显示与缩放，滚动或者移动鼠标时都会重新计算一次
  - 元素适当定义高宽，避免回流
  - 给图片设置尺寸大小，否则在载入时从0->1的过程会导致回流
  - 不适用table布局
  - 尽量使用CSS实现效果，避免使用JS实现，CSS效率更好专业的东西做专业的事情
- 渲染层
  - 频繁绘制的DOM或动画，使用绝对定位fix，absolute脱离文档流，新建render layer
  - 对于进行动画的元素，使用硬件渲染(transform: translateZ(z))
- DOM优化
  - 频繁操作时缓存DOM
  - 减少DOM深度及DOM数量，数量越多层级越深解析花费时间就越多，尽量保持DOM元素简洁和层级较少
  - 批量操作DOM，如使用innerHTML进行拼接(注意XSS攻击)
  - 批量操作CSS样式，尽量使用class，如果是style，则尽量使用style.cssText=''
  - 在内存中操作DOM，使用DocumentFragment
  - 离线操作DOM，操作大量DOM的情况下，可先使用display: none;先进行隐藏
  - 分离读写操作。浏览器具有惰性机制，连续多次的操作DOM会维护一个flush队列
  - 事件代理。利用事件冒泡机制将监听事件注册在父级元素上，页面的监听事件数量和页面性能呈现反比趋势
    - 降低内存消耗
    - 动态添加子节点不必起监听事件
  - 防抖&节流
  - 及时清理内存环境
    - 及时消除对象引用
    - 清除定时器
    - 清除事件监听器
    - 创建最小作用与变量
    - 谨慎使用闭包，并及时清理

6.性能更好的API

- 用对选择器：选择器种类: id选择器，类选择器，标签选择器，属性选择器，相邻选择器，子类选择器，后代选择器，通配符选择器，伪类选择器，伪元素选择器
- rAF：代替settTimeout、setInterval
- IntersectionOberser：不会造成页面回流
- 使用web worker：计算密集型任务可以交给worker线程，不影响主线程

7.webpack性能优化

- 打包公共代码
- 剔除无用代码：tree shaking
  - JS的tree shaking主要通过uglifyjs插件来完成
  - CSS的tree shaking主要通过purify CSS来实现
- 长缓存优化
  - 将hash替换为chunkhash，当chunk不变，缓存依然有效
  - 使用name而不是id
- 公用代码内联：使用html-webpack-inline-chunk-plugin插件将mainfest.js内联到html文件中