2018/1/22

# 慕课——web前端性能优化<https://coding.imooc.com/class/130.html>

## 第二章

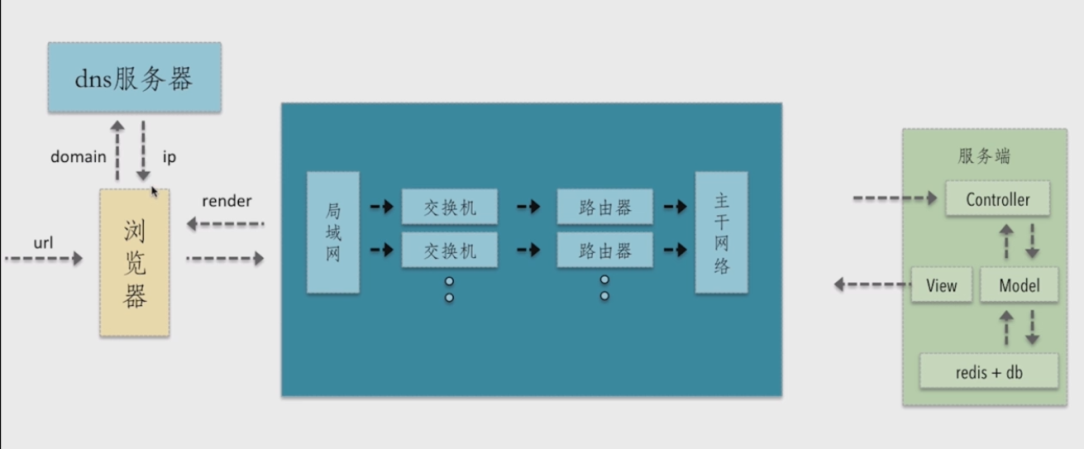
1. 优化要点：
2. 减少http请求数量（资源的合并）
3. 减少请求资源的大小（资源的压缩）
4. Web前端的本质是一种GUI软件

注意：web前端与普通GUI软件的区别

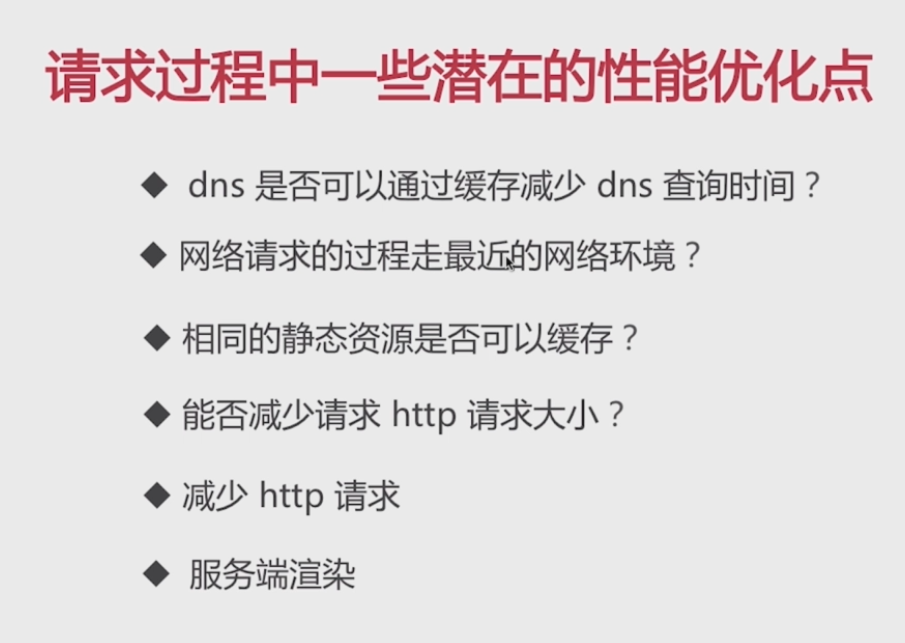
CS架构，例如手机app，下载app到本地安装后，软件运行时实际访问的是本地的资源。

BS架构，将产品发布在远程服务器，用户访问时，访问的是远程服务器的资源。

1. 浏览器的一个请求从发送到返回都经历了什么？



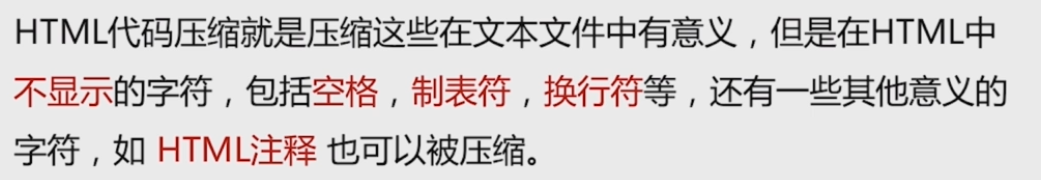
1. 性能优化点



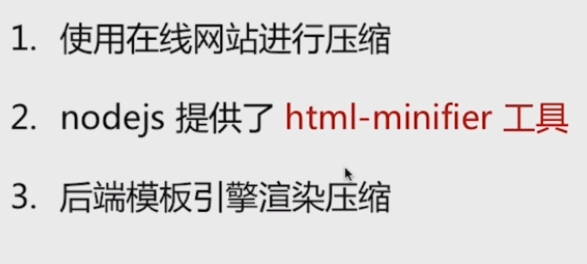
**深入理解http的请求过程是前端性能优化的关键点，这里的深入理解是基于项目业务及使用的技术栈的深入理解。**

2-2.资源的合并与压缩

1. （1）Html压缩



（2）如何压缩



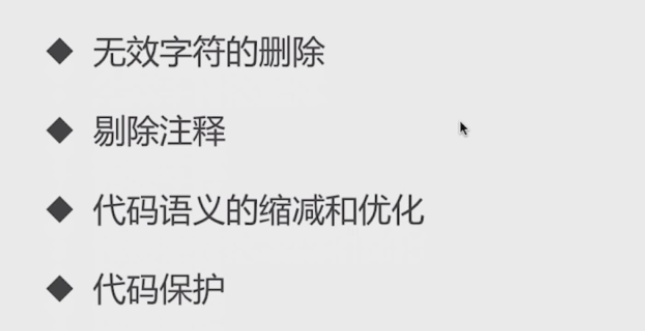
1. CSS压缩

（1）删除无效代码，语义合并。

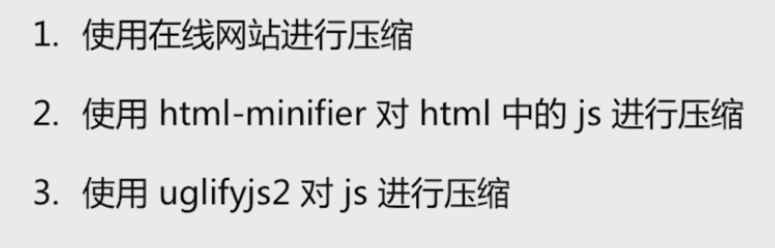
（2）如何压缩。



1. js压缩与混乱

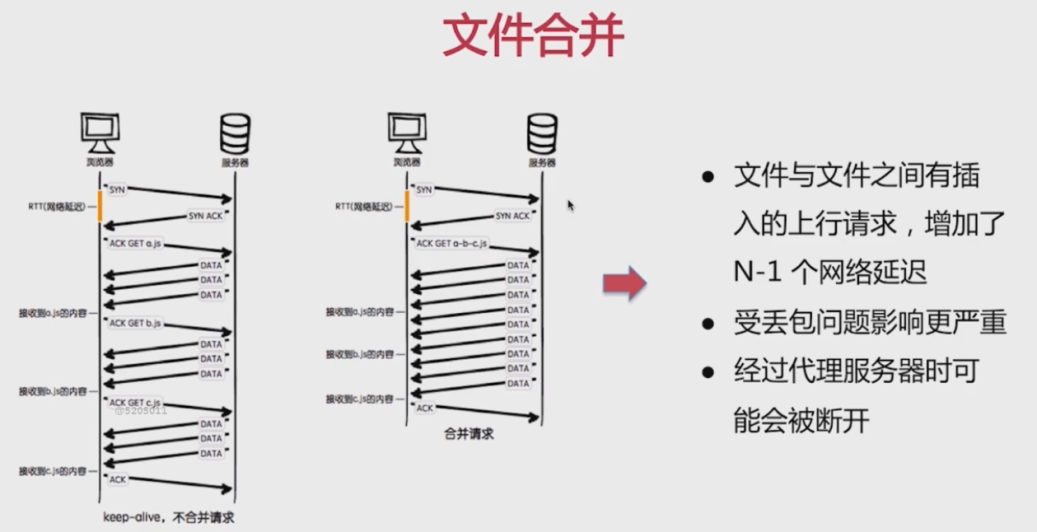


（2）如何压缩



1. 文件合并

（1）文件不合并存在的问题



（2）文件合并的缺点（灵活应变）

A．首屏渲染的问题

当页面的内容是依赖与js文件来加载渲染时（eg：使用react或是vue框架实现商品展示页面，假设在a.js中），如果把a.js，b.js，c.js 合并成一个 abc.js ,那么abc.js会很大，虽然请求的次数由三次变为一次，但是请求的资源变大了，因为之前只要加载a.js。——不同页面的js文件的单独打包。

B．缓存失效的问题

Md5戳。a.js，b.js，c.js中任意代码区域的改变都会导致abc.js文件的重新加载。——解决方式：公共库的合并，业务代码不合并。

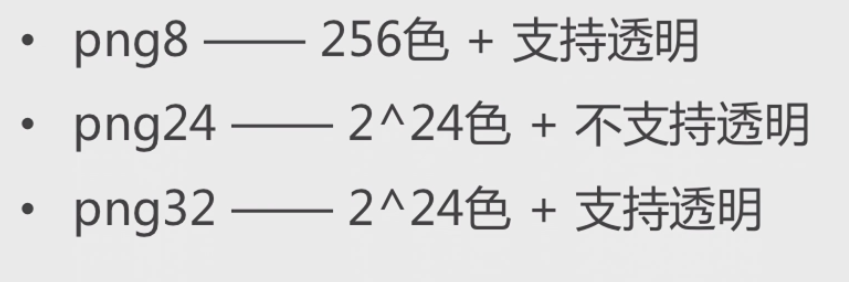
1. 实战

注意：

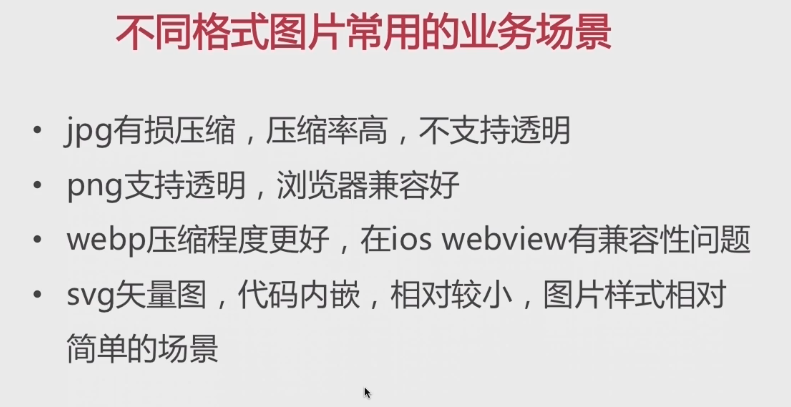
1. 浏览器同一域名请求静态资源并发请求限制。（有上限）
2. 项目大时一定要做性能优化。
3. Web前端：动态增量加载静态资源。（受网络环境的影响）

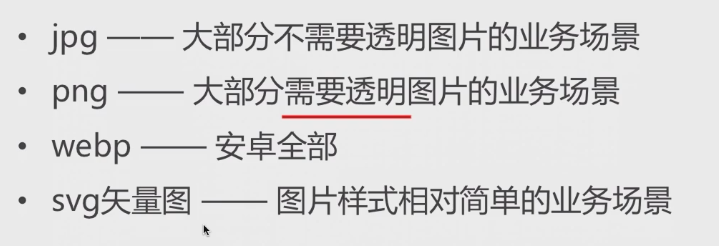
## **第三章** 图片相关的性能优化

1. png图片



2. 不同格式图片的业务场景





3. 图片优化

（1）雪碧图

（2）Image inline：减少网络请求次数。

（3）矢量图：能用矢量图替代的尽量用矢量图。

（4）安卓下使用webp

4. 图片压缩网站

（1）<https://tinypng.com/>

（2）<http://zhitu.isux.us/> （webp）

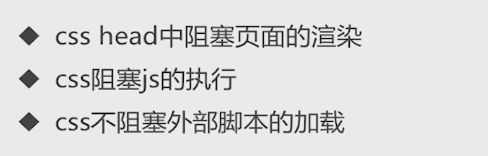
## 第四章 css和js的装载与执行

**核心：一个网站在浏览器中是如何渲染的?**

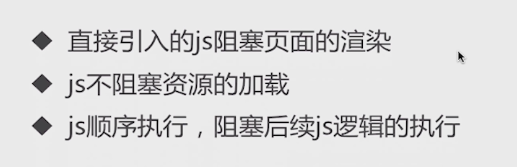
1. 词法分析
2. 单个域名浏览器的并发数是受限制的——设置多个CDN域名
3. CSS阻塞

注意：

1. head中用link标签引入css会阻塞页面的渲染，这样可以避免用户打开页面时，没有样式的情况。
2. css的加载会阻塞js代码的执行，但是不会阻塞加载，因为js代码中可能存在基于当前css样式的事件。（eg：js改变字体颜色）



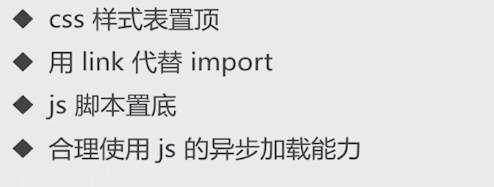
1. JS阻塞（预加载）



1. 依赖关系



1. JS引入方式：
2. 直接引入——阻塞页面渲染（script标签）
3. defer——不会阻塞页面渲染（执行时所有的dom树都渲染完成，顺序执行，可以保证依赖关系。）
4. async——不会阻塞页面渲染（因为不会阻塞页面的渲染，可以更快的展示页面，但是前提是js代码对页面没有展示上的影响，不保证按引入的顺序执行，没有依赖关系，立即执行。）
5. 异步动态引入js
6. 加载和执行的一些优化点



1. 动态异步引入js



1. 使用@import，link引入css

（1）@import CSS模块化。

（2）@import不支持并发执行。（chrome现在浏览器已支持）

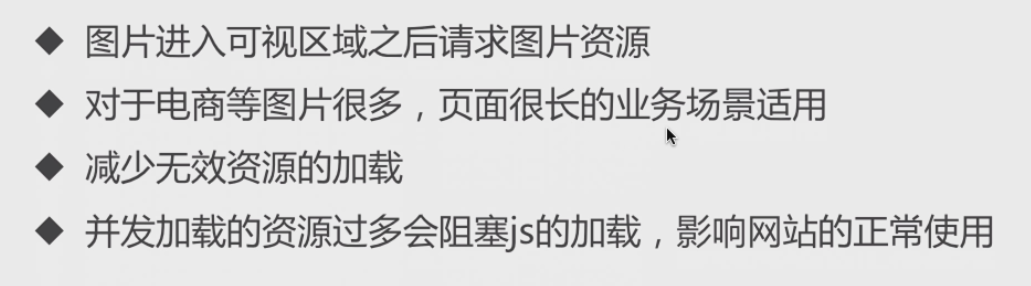
（3）@import页面加载完成完再执行css代码。（chrome现在浏览器已支持）

（4）@import 层数不能太深。

eg: 在a.css 中@import b.css，是先加载a.css完成后再加载b.css。

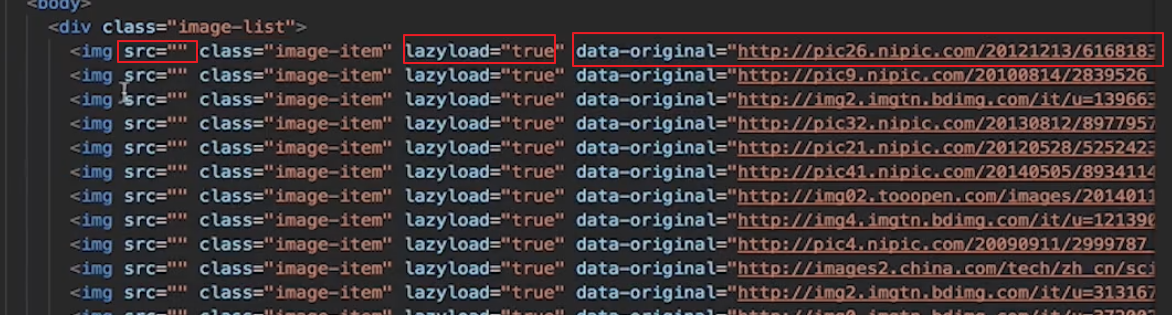
## 第五章 懒加载和预加载

1. 懒加载（把data-image中的地址放到background的url中）



1. 懒加载实现

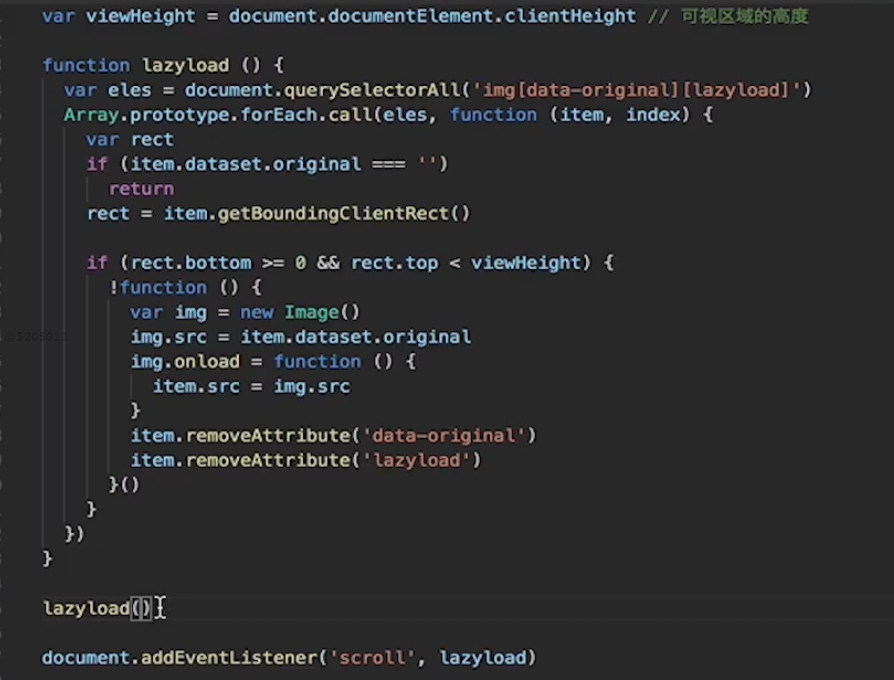
Html



JS

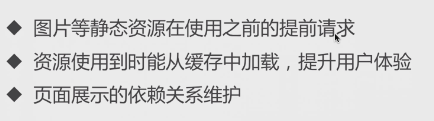
注意：（1）首屏要先调用一次lazyload。

（2）图片一定要占位。



B． zepto.lazyload

2. 预加载



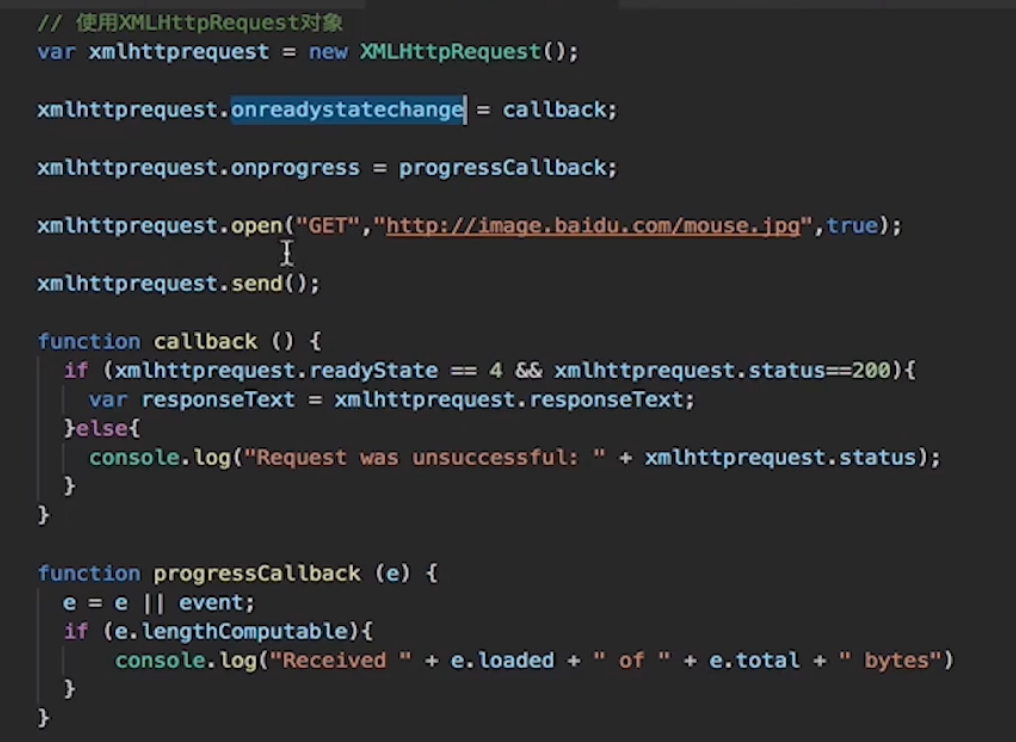
1. 使用image标签将图片加载进来，但是不显示：display：none。
2. 使用Image对象



1. 使用XHR

注意：

1. 存在跨域问题
2. 优点：更加精确的控制整个预加载的过程。



1. 相关库实现： <http://www.createjs.cc/preloadjs/>

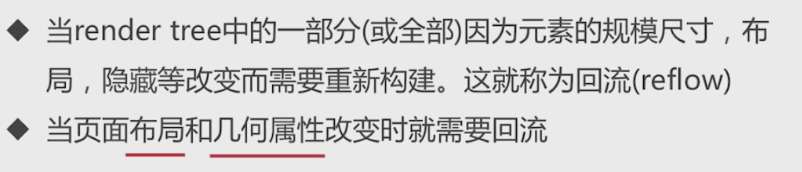


## 第六章 重绘与回流

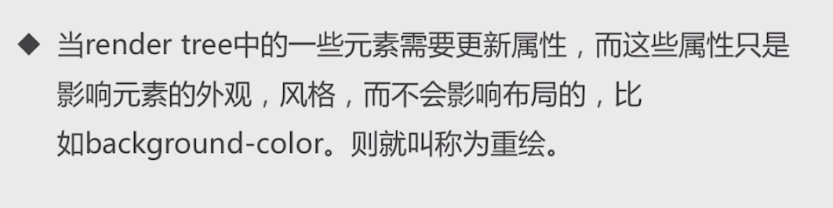
1.



2. 回流

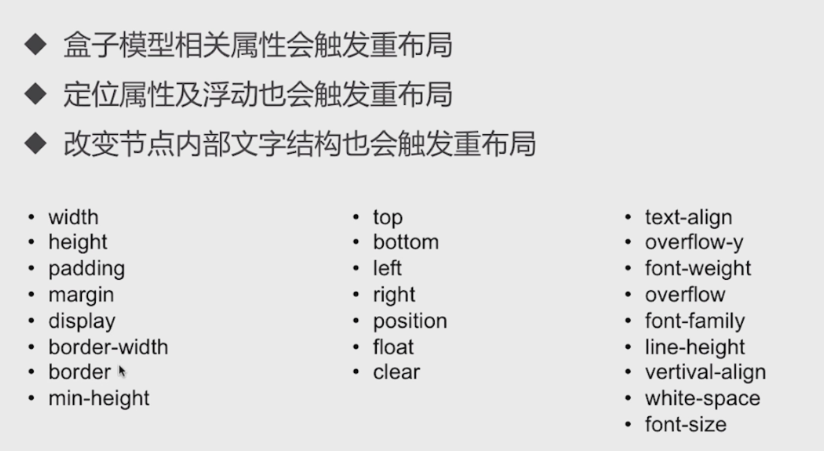


3. 重绘



注意：回流必定会引起重绘，但是重绘不一定会引起回流。

4. 触发页面重新布局的属性——回流

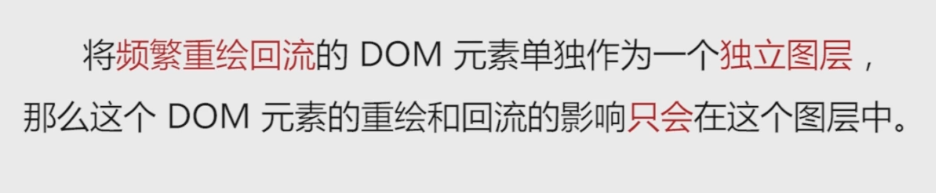


1. 触发页面重绘



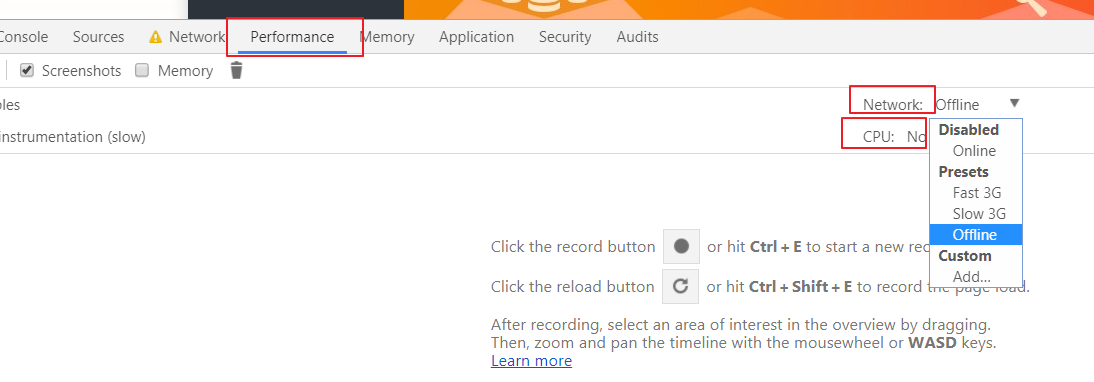
1. 避免重绘回流



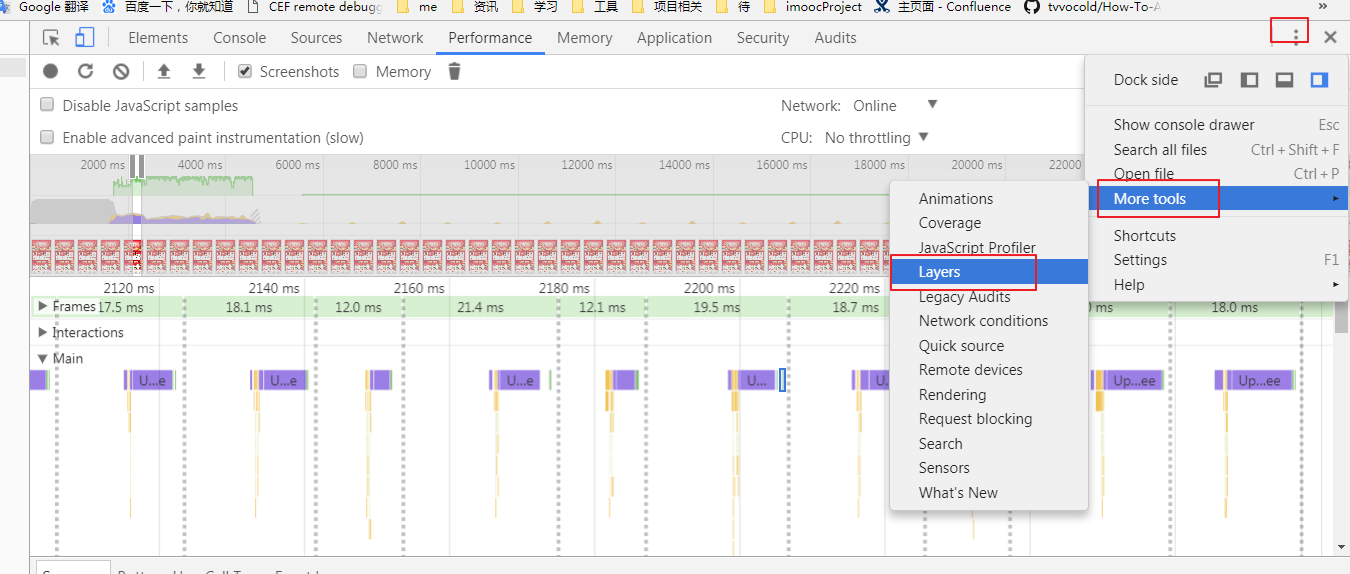


？如何新建图层（另了解）

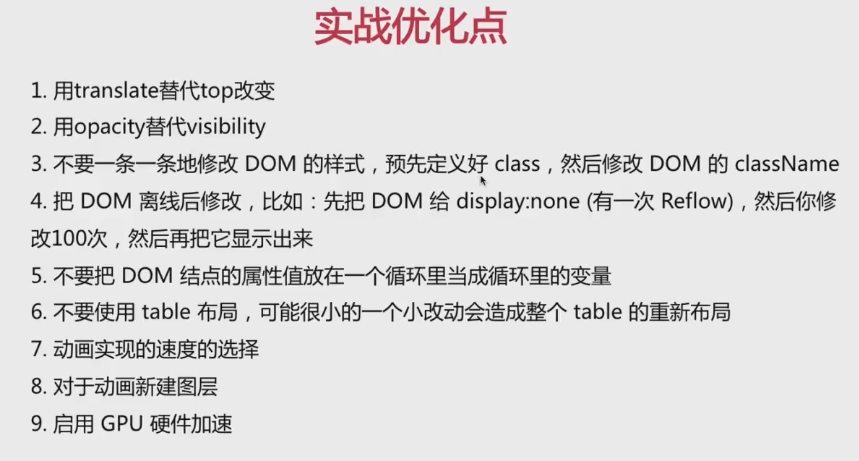
Chrome浏览器**performance**工具测试网页性能（回流及重绘性能查看）



图层查看Layers



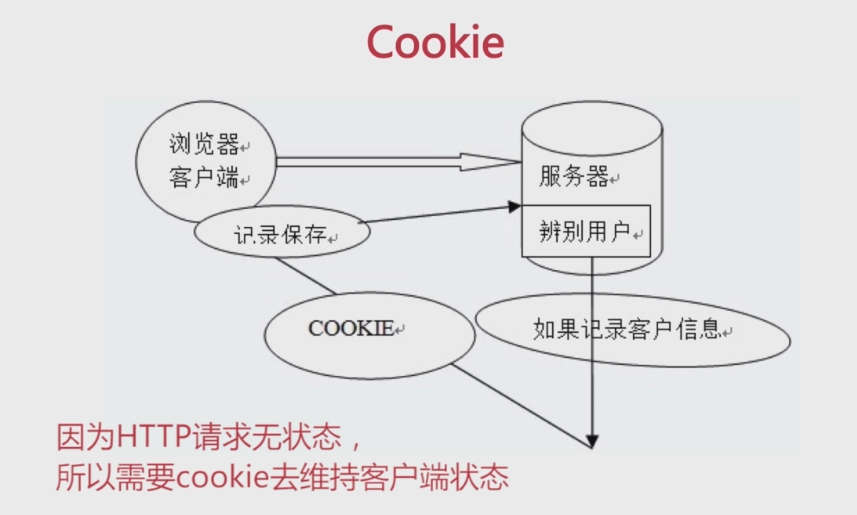
1. 实战优化点



## 第七章 浏览器存储

1. Cookie

（1）



（2）cookie生成

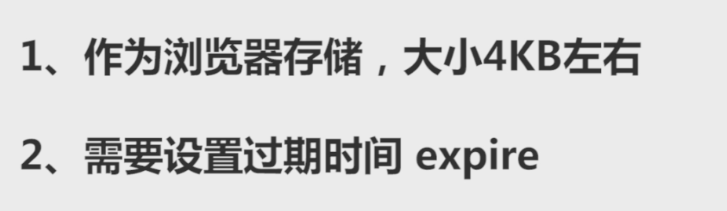
A. 用于浏览器端与客户端的交互

服务端生成：http response header 中的 set-cookie。

B. 用于客户端自身数据的存储

客户端：js中可以通过document.cookie 读写cookie。

1. cookie存储的限制

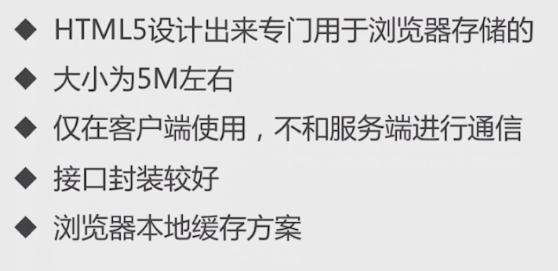


1. cookie重要属性：httponly（js不能读写）。
2. cookie是一个域名维度下的get

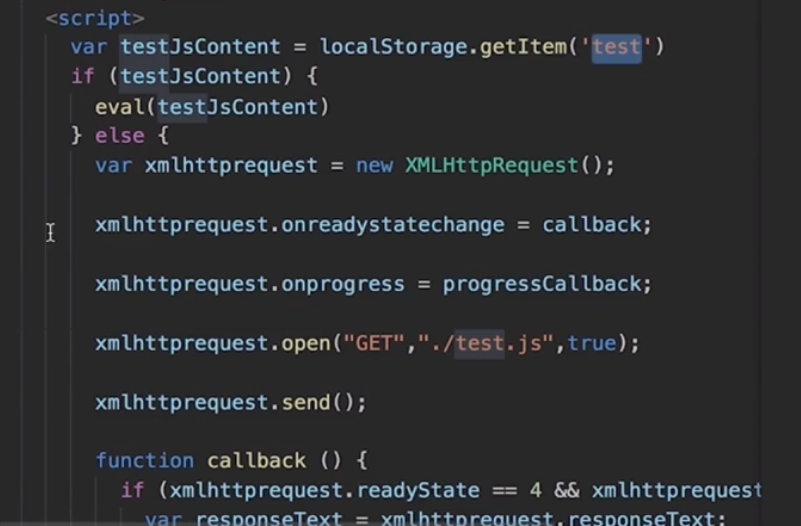
cookie中在相关域名下面——cdn的流量损耗。

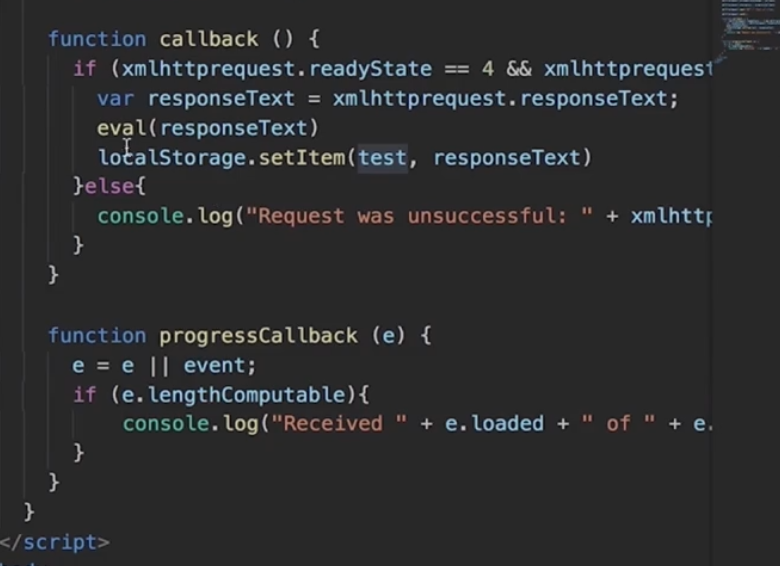
解决方案：cdn的域名和主站的域名要分开。（百度是这么做的）

2. LocalStorage

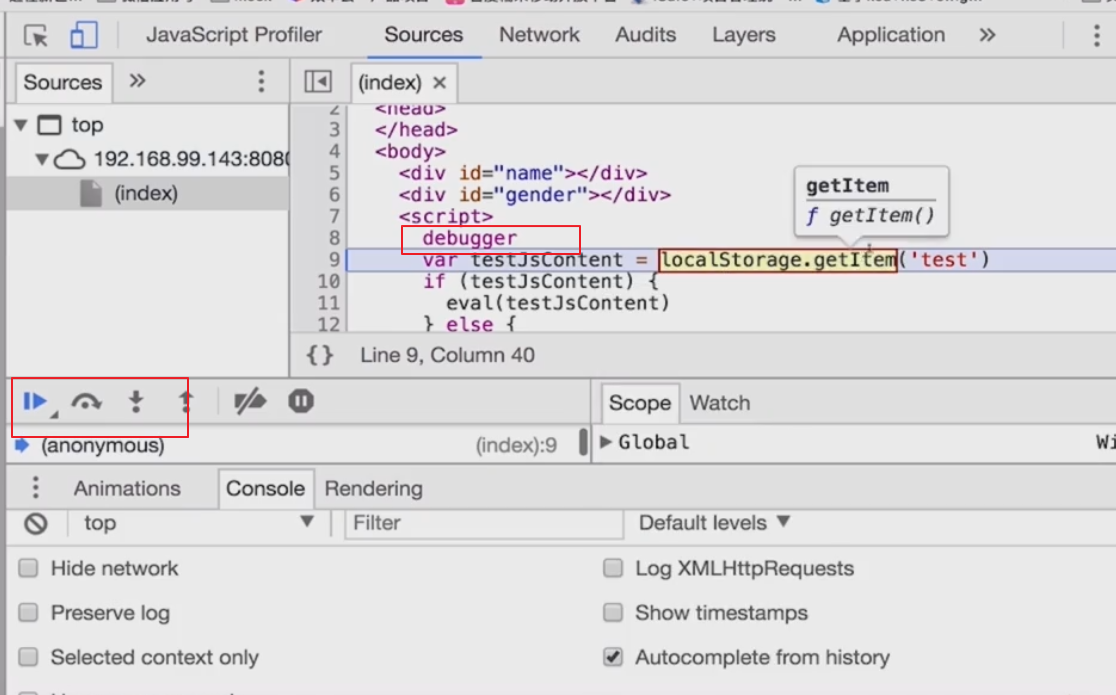


1. 使用localStorage缓存js文件





（2）在js代码打断点调试



3． SessionStorage

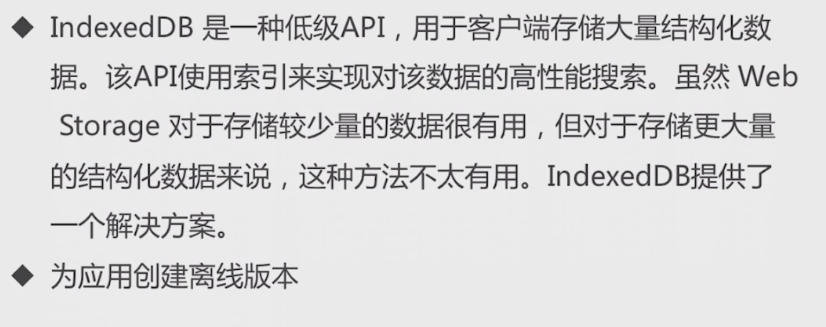
理解会话：浏览器中的一个Tab标签页就是一个会话。



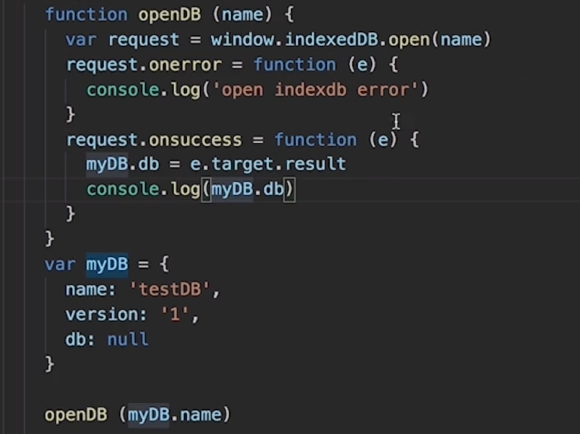
应用场景：

1. 表单页面，用户填写到一半刷新了页面，这时候之前填的内容会清空，可以用SessionStorage存储相关的数据，再进行处理。（用户关闭标签结束会话）
2. 一个表单在进行多页面切换的时候数据的传递。（表单又多页面组成：eg：填写姓名工号一个页面——填写详细信息一个页面）

4. IndexedDB

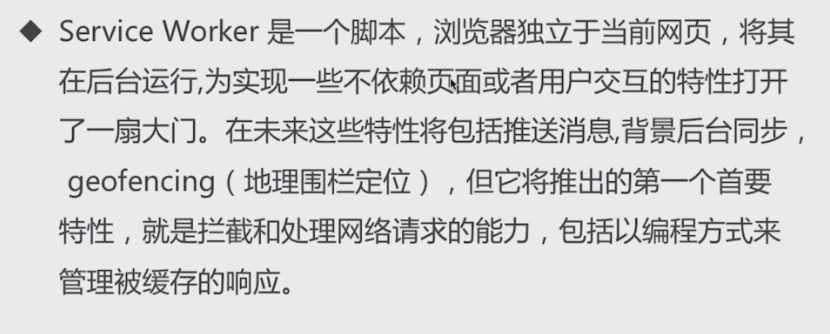


1. 建立打开indexDB

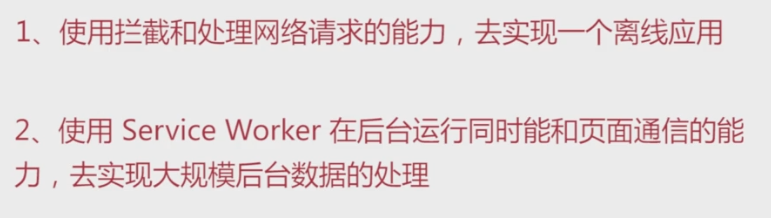


（2）其他操作在使用时另行学习~

1. Service Workers
2. 可以理解在浏览器JS主线程之外的另外一个线程。JS主线程可以将一些耗时较长，同时又不依赖于页面和用户交互的特性放到Service Workers在浏览器的后台执行。



1. Server Workers 应用点

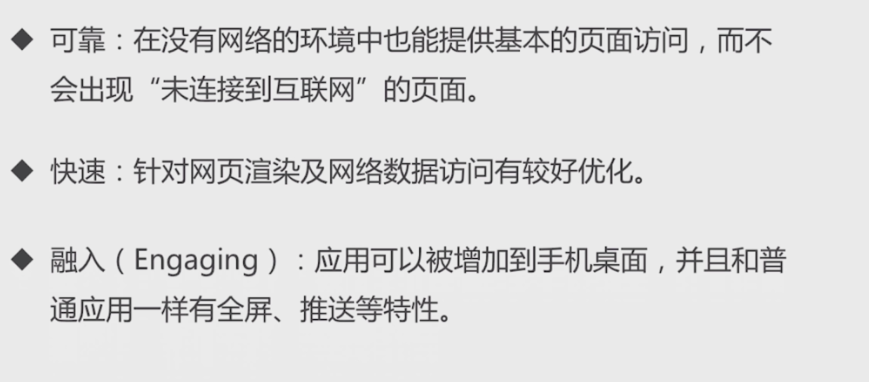


1. PWA

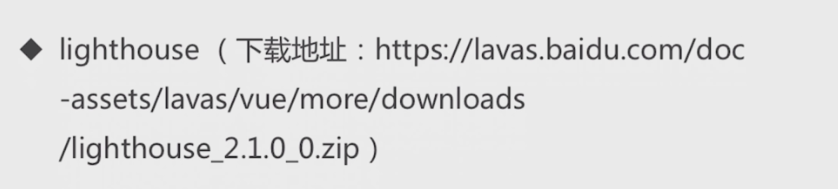
（1）渐进式理解：手机web页面在离线，弱网，网络环境正常等情况下的体验是否有渐进式的体验。



（2）PWA标准



（3）谷歌用于检测网站是否是PWA还有网站性能的工具（chrome插件，需要翻墙，下面的地址是老师的）



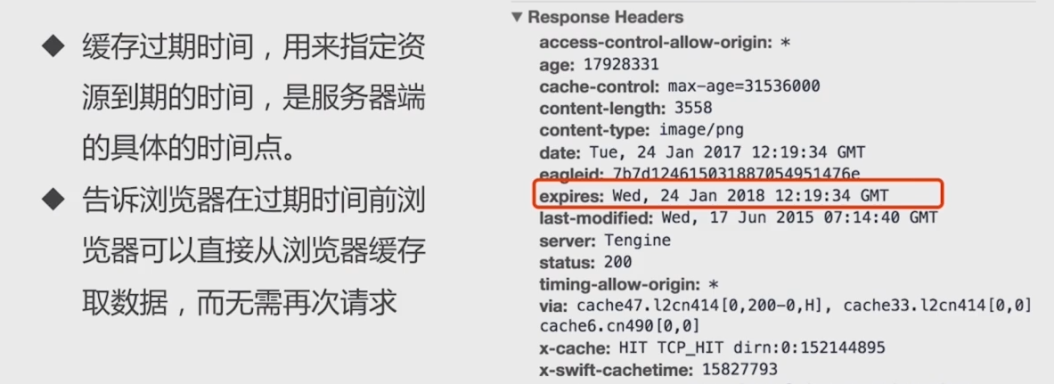
## 第八章 缓存（建立合作化的基于客户端和浏览器端协商的缓存机制）

查看Chrome浏览器的缓存：chrome://cache

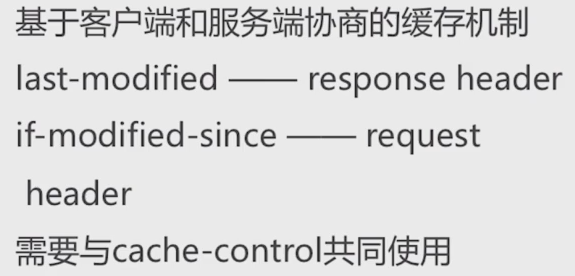
1. Cache-Control

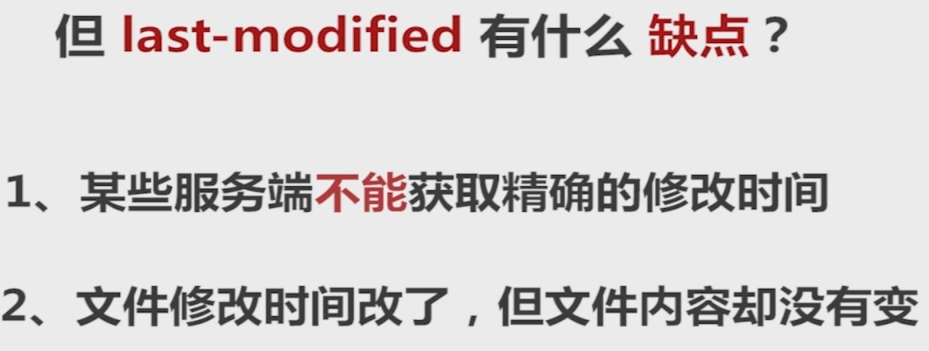


2. Expires（优先级比max-age低）



3. Last-Modified/If-Modified-Since





4.

