前端全链路性能优化实战

极客时间

扫码试看/订阅

《前端全链路性能优化实战》视频课程

第一章：课程介绍和内容综述

1.1 课程介绍

1.2 内容综述

第二章：静态资源优化

2.1 图片格式和应用场景介绍

2.2 图片优化细则

2.3 图片服务器自动优化解密

2.4 HTML 优化细则

2.5 CSS 优化细则

2.6 JavaScript 优化细则

2.7 JavaScript 缓存优化

2.8 JavaScript 模块化加载方案和选型

第二章：静态资源优化

2.9 减少回流和重绘重要举措

2.10 DOM 编程优化的方式方法

2.11 静态文件压缩工具介绍

2.12 静态文件打包方案

2.13 静态文件版本号更新策略

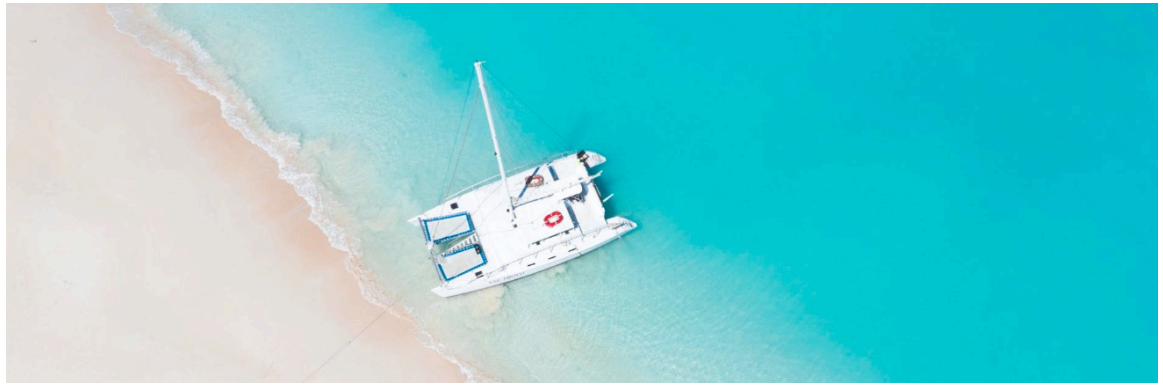
2.14 前端构建工具介绍和选型建议

2.15 webpack 打包优化

2.1 图片格式和应用场景介绍

JPEG（Joint Photographic Experts Group）

- 联合图像专家小组是一种针对彩色照片而广泛使用的有损压缩图形格式。
  - 介绍：栅格图形。常用文件扩展名为 .jpg，也有 .jpeg、.jpe。JPEG 在互联网上常被应用于存储和传输照片。
  - 不适合：线条图形和文字、图标图形，因为它的压缩算法不太这些类型的图形；并且不支持透明度。
  - 非常适合：颜色丰富的照片、彩色图大焦点图、通栏 banner 图；结构不规则的图形。



PNG（Portable Network Graphics）

- 便携式网络图形是一种无损压缩的位图图形格式，支持索引、灰度、RGB 三种颜色方案以及 Alpha 通道等特性。
  - 介绍：栅格图形。PNG 最初是作为替代 GIF 来设计的，能够显示 256 色，文件比 JPEG 或者 GIF 大，但是 PNG 非常好的保留了图像质量。支持 Alpha 通道的半透明和透明特性。最高支持 24 位彩色图像（PNG-24）和 8 位灰度图像（PNG-8）。
  - 不适合：由于是无损存储，彩色图像体积太大，所以不太适合。
  - 非常适合：纯色、透明、线条绘图，图标；边缘清晰、有大块相同颜色区域；颜色数较少但需要半透明。

GIF（Graphics Interchange Format）

- 图像互换格式是一种位图图形文件格式，以 8 位色（即 256 种颜色）重现真彩色的图像，采用 LZW 压缩算法进行编码。
  - 介绍：栅格图形。支持 256 色；仅支持完全透明和完全不透明；如果需要比较通用的动画，GIF 是唯一选择。
  - 不适合：每个像素只有 8 比特，不适合存储彩色图片。
  - 非常适合：动画，图标。

Webp

- Webp 是一种现代图像格式，可为图像提供无损压缩和有损压缩，这使得它非常灵活。由 Google 在购买 On2 Technologies 后发展出来，以 BSD 授权条款发布。
  - 介绍：优秀算法能同时保证一定程序上的图像质量和比较小的体积；可以插入多帧，实现动画效果；可以设置透明度；采用 8 位压缩算法。无损的 Webp 比 PNG 小 26%，有损的 Webp 比 JPEG 小 25-34%，比 GIF 有更好的动画。
  - 不适合：最多处理 256 色，不适合于彩色图片。
  - 非常适合：适用于图形和半透明图像。

2.2 图片优化细则

用工具进行图片压缩

- 压缩 png
- node-pngquant-native
- 跨平台，压缩比高，压缩 png24 非常好。
- 说明文档：
  - https://www.npmjs.com/package/node-pngquant-native
- 安装方法：
  - npm install node-pngquant-native

用工具进行图片压缩

- 压缩 jpg
- jpegtran
- 跨平台，有 Linux 、Mac、 Windows 的解决方案
- 官网：http://jpegclub.org/jpegtran/
- 安装方法：npm install -g jpegtran
- 使用方法：
  - jpegtran -copy none -optimize -outfile out.jpg in.jpg

用工具进行图片压缩

- 压缩 gif
- Gifsicle：通过改变每帧比例，减小 gif 文件大小，同时可以使用透明来达到更小的文件大小，目前公认的解决方案。
- 安装：
  - http://www.lcdf.org/gifsicle/
- 使用方式：
  - 优化级别设置为不小于 2，1 的话基本不压缩 gifsicle --optimize=3 -o out.gif in.gif
  - 将透明部分截去 gifsicle --optimize=3 --crop-transparency -o out.gif in.gif

图片尺寸随网络环境变化

- 不同网络环境（Wifi/4G/3G）下，加载不同尺寸和像素的图片，通过在图片 URL 后缀加不同参数改变。

- http://img13.360buyimg.com/n1/s100x100_jfs/t2443/71/2538811251/470889/c2ec38b3/570f3438N81a4b62c.jpg

响应式图片

- JavaScript 绑定事件检测窗口大小
- CSS 媒体查询

```
@media screen and (max-width:640px) {
        my_image{ width:640px; }
}
```

- img 标签属性

```
<img srcset="img-320w.jpg, img-640w.jpg 2x, img-960w.jpg 3x"
        src=“img-960w.jpg” alt=“img”>
```

（x 描述符：表示图像的设备像素比）

逐步加载图像

- 使用统一占位符
- 使用 LQIP
  - 低质量图像占位符（Low Quality Image Placeholders）
  - 安装：npm install lqip
  - 源码：https://github.com/zouhir/lqip-loader
- 使用 SQIP
  - 基于 SVG 的图像占位符（SVG Quality Image Placeholders）
  - 安装：npm install sqip
  - 源码：https://github.com/axe312ger/sqip

真的需要图片吗？

- Web Font 代替图片
- 使用 Data URI 代替图片
- 采用 Image spriting（雪碧图）

2.3 图片服务器自动优化解密

图片服务器自动优化解密

极客时间

- 名词解释
  - 图片服务器自动化优化是可以在图片 URL 链接上增加不同特殊参数，服务器自动化生成。
  - 不同格式、大小、质量的图片。
- 处理方式
  - 图片裁剪：按长边、短边、填充、拉伸等缩放。
  - 图片格式转换：支持 JPG，GIF，PNG，WebP 等，支持不同的图片压缩率。
  - 图片处理：添加图片水印、高斯模糊、重心处理、裁剪边框等。
  - AI 能力：鉴黄以及智能抠图、智能排版、智能配色、智能合成等 AI 功能。

图片服务器自动优化解密

- 默认 jpg
  - https://m.360buyimg.com/test/s500x500_jfs/t2362/199/2707005502/100242/616257ce/56e66b21N7b8c2be8.jpg
- 大小 100*100 的 jpg
  - https://m.360buyimg.com/test/s100x100_jfs/t2362/199/2707005502/100242/616257ce/56e66b21N7b8c2be8.jpg
- webp 格式的图片
  - https://m.360buyimg.com/test/s500x500_jfs/t2362/199/2707005502/100242/616257ce/56e66b21N7b8c2be8.webp
- 质量压缩至10%
  - https://m.360buyimg.com/test/s500x500_jfs/t2362/199/2707005502/100242/616257ce/56e66b21N7b8c2be8.jpg!q10

2.4 HTML 优化细则

精简 HTML 代码

- 减少 HTML 的嵌套
- 减少 DOM 节点数
- 减少无语义代码（比如: 消除浮动）
- 删除 http 或者 https，如果URL的协议头和当前页面的协议头一致的，或者此 URL 在多个协议头都是可用的，则可以考虑删除协议头
- 删除多余的空格、换行符、缩进和不必要的注释
- 省略冗余标签和属性
- 使用相对路径的 URL

文件放在合适位置

- CSS 样式文件链接尽量放在页面头部
  - CSS 加载不会阻塞 DOM tree 解析，但是会阻塞 DOM Tree 渲染，也会阻塞后面 JS 执行。任何 body 元素之前，可以确保在文档部分中解析了所有 CSS 样式（内联和外联），从而减少了浏览器必须重排文档的次数。如果放置页面底部，就要等待最后一个 CSS 文件下载完成，此时会出现"白屏"，影响用户体验。
- JS 引用放在 HTML 底部
  - 防止 JS 的加载、解析、执行对阻塞页面后续元素的正常渲染。

增强用户体验

- 设置 favicon.ico
  - 网站如果不设置 favicon.ico，控制台会报错，另外页面加载过程中也没有图标 loading 过程，同时也不利于记忆网站品牌，建议统一添加。
- 增加首屏必要的 CSS 和 JS
  - 页面如果需要等待所的依赖的 JS 和 CSS 加载完成才显示，则在渲染过程中页面会一直显示空白，影响用户体验，建议增加首屏必要的 CSS 和 JS，比如页面框架背景图片或者 loading 图标，内联在 HTML 页面中。这样做，首屏能快速显示出来，相对减少用户对页面加载等待过程。（比如新浪微博 M 站页面框架）

2.5 CSS 优化细则

提升 CSS 渲染性能

极客时间

- 提升 CSS 渲染性能
  - 谨慎使用 expensive 属性
    - 如:nth-child 伪类；position: fixed 定位
  - 尽量减少样式层级数
    - 如div ul li span i {color: blue;}
  - 尽量避免使用占用过多 CUP 和内存的属性
    - 如text-indnt:-99999px
  - 尽量避免使用耗电量大的属性
    - 如CSS3 3D transforms、CSS3 transitions、Opacity

合适使用 CSS 选择器

- 尽量避免使用 CSS 表达式
  - background-color: expression((new Date()).getHours()%2 ? "#FFF" : "#000");
- 尽量避免使用通配选择器
  - body > a {font-weight:blod;}
- 尽量避免类正则的属性选择器
  - *=， |=， ^=， $=

提升 CSS 文件加载性能

- 使用外链的 CSS
- 尽量避免使用 @import

精简 CSS 代码

- 使用缩写语句
- 删除不必要的零
- 删除不必要的单位，如px
- 删除除过多分号
- 删除空格和注释
- 尽量减少样式表的大小

合理使用 Web Fonts

- 将字体部署在 CDN 上
- 将字体以 base64 形式保存在 CSS 中并通过 localStorage 进行缓存
- Google 字体库因为某些不可抗拒原因，应该使用国内托管服务

CSS 动画优化

- 尽量避免同时动画
- 延迟动画初始化
- 结合 SVG

2.6 JavaScript 优化细则

JavaScript 优化总体原则

- 当需要时才优化
- 考虑可维护性

提升 JavaScript 文件加载性能

- 加载元素的顺序 CSS 文件放在 <head> 里， JavaScript 文件放在 <body> 里。

JavaScript 变量和函数优化

- 尽量使用 id 选择器
- 尽量避免使用 eval
- JavaScript 函数尽可能保持简洁
- 使用事件节流函数
- 使用事件委托

JavaScript 动画优化

- 避免添加大量 JavaScript 动画
- 尽量使用 CSS3 动画
- 尽量使用 Canvas 动画
- 合理使用 requestAnimationFrame 动画代替 setTimeout、setInterval
  - requestAnimationFrame可以在正确的时间进行渲染，setTimeout（callback）和 setInterval（callback）无法保证 callback 回调函数的执行时机

合理使用缓存

- 合理缓存 DOM 对象
- 缓存列表长度
- 使用可缓存的 Ajax

2.7 JavaScript 缓存优化

Cookie

- 通常由浏览器存储，然后将 Cookie 与每个后续请求一起发送到同一服务器。收到 HTTP 请求时，服务器可以发送带有 Cookie 的 header 头。可以给 Cookie 设置有效时间。

- 应用于：

  - 会话管理：登录名，购物车商品，游戏得分或服务器应要记录的其他任何内容
  - 个性化：用户首选项，主题或其他设置
  - 跟踪：记录和分析用户行为，比如埋点

sessionStorage

- 创建一个本地存储的键/值对
- 应用于：
  - 页面应用页面之间传值

IndexedDB

- 索引数据库
- 应用于：
  - 客户端存储大量结构化数据
  - 没有网络连接的情况下使用（比如 Google Doc、石墨文档）
  - 将冗余、很少修改、但经常访问的数据，以避免随时从服务器获取数据

LocalStorage

- 本地存储
- 应用于：
  - 缓存静态文件内容 JavaScript /CSS（比如百度M站首页）
  - 缓存不常变更的 API 接口数据
  - 储存地理位置信息
  - 浏览在页面的具体位置

2.8 JavaScript 模块化加载方案和选型

JavaScript 模块化加载方案和选型

- CommonJS
  - 旨在 Web 浏览器之外为 JavaScript 建立模块生态系统
  - Node.js 模块化方案受 CommonJS
- AMD (Asynchronous Module Definition)（异步模块定义）规范
  - RequireJS 模块化加载器：基于 AMD API 实现
- CMD（ Common Module Definition）（通用模块定义）规范
  - SeaJS 模块化加载器：遵循 CMD API 编写
- ES6 import

2.9 减少回流和重绘重要举措

CSS

- 避免过多样式嵌套
- 避免使用 CSS 表达式
- 使用绝对定位，可以让动画元素脱离文档流
- 避免使用 table 布局
- 尽量不使用 float 布局
- 图片最好设置好 width 和 height
- 尽量简化浏览器不必要的任务，减少页面重新布局
- 使用 Viewport 设置屏幕缩放级别
- 避免频繁设置样式，最好把新 style 属性设置完成后，进行一次性更改
- 避免使用引起回流/重绘的属性，最好把相应变量缓存起来

JavaScript

- 最小化回流和重排
  - 为了减少回流发生次数，避免频繁或操作 DOM，可以合并多次对 DOM 修改，然后一次性批量处理。
- 控制绘制过程和绘制区域
  - 绘制过程开销比较大的属性设置应该尽量避免减少使用
  - 减少绘制区域范围

2.10 DOM 编程优化的方式方法

控制 DOM 大小

- 众所周知，页面交互卡顿和流畅度很大一部分原因就是页面有大量 DOM 元素。想象一下，从一个上万节点的 DOM 树上，使用 querySelectorAll 或 getElementByTagName 方法查找某一个节点，是非常耗时的。另外元素绑定事件，事件冒泡和事件捕获的执行也会相对耗时。

- 通常控制 DOM 大小的技巧包括：
  - 合理的业务逻辑
  - 延迟加载即将呈现的内容

简化 DOM 操作

- 对DOM节点的操作统一处理后，再统一插入到 DOM Tree中。
- 可以使用 fragment，尽量不在页面 DOM Tree 里直接操作。
- 现在流行的框架 Angular、React、Vue 都在使用虚拟 DOM 技术，通过 diff 算法简化和减少 DOM 操作。

2.11 静态文件压缩工具介绍

静态文件压缩工具介绍

- HTML 压缩工具
  - html-minifier https://www.npmjs.com/package/html-minifier
- CSS 压缩工具
  - clean-css https://www.npmjs.com/package/clean-css
- JavaScript 压缩工具：
  - uglify-js https://www.npmjs.com/package/uglify-js
  - 使用方法：uglifyjs in.js -o out.js

2.12 静态文件打包方案

静态文件打包方案

- 公共组件拆分
- 压缩： JavaScript /CSS/图片
- 合并： JavaScript /CSS 文件合并，CSS Sprite
- Combo： JavaScript /CSS 文件 Combo http://cdn.com/??a.js,b.js 内容

2.13 静态文件版本号更新策略

静态文件版本号更新策略

- 缓存更新
  - CDN 或 ng 后台刷新文件路径，更新文件header头
- 文件 name.v1-v100.js
  - 大功能迭代每次新增一个大版本，比如由 v1 到 v2
  - 小功能迭代新增加 0.0.1 或者 0.1.0，比如从 v1.0.0 至 v1.0.1
  - 年末 ng 统一配置所有版本 302 至最新版

静态文件版本号更新策略

- 时间戳.文件 name.js
  - 以每次上线时间点做差异
- 文件 hash.文件 name.js
  - 以文件内容 hash 值做 key
  - 每次上线，文件路径不一致

2.14 前端构建工具介绍和选型建议

前端构建工具介绍和选型建议

- Grunt
  - 最早，一个项目需要定制多个小任务和引用多个插件（质量参差不齐）
- Gulp
  - 通过流（Stream）来简化多个任务间的配置和输出，配置代码相对较少
- Webpack
  - 预编译，中间文件在内存中处理，支持多种模块化，配置相对很简单
- FIS
- JDF

2.15 webpack 打包优化

webpack 打包优化

- 定位体积大的模块
- 删除没有使用的依赖
- 生产模式进行公共依赖包抽离
- 开发模式进行 DLL & DllReference 方式优化

极客时间

扫码试看/订阅

《前端全链路性能优化实战》视频课程