

前端性能优化

前端性能优化

1. 课前准备
2. 课堂主题
3. 课堂目标
4. 知识点

从输入 URL 到页面加载完成，发生了什么？

DNS

TCP

三次握手

HTTP

浏览器缓存机制

http Cache

memory Cache

Service Worker Cache

push cache

文件打包

图片优化

gzip

本地 存储

CDN

服务端渲染

vue服务端渲染

nuxt.js 服务端渲染框架体验

react服务端渲染

1. 课前准备

2. 课堂主题

1. 性能优化宏观概念
2. 代码少执行
3. 缓存

3. 课堂目标

4. 知识点

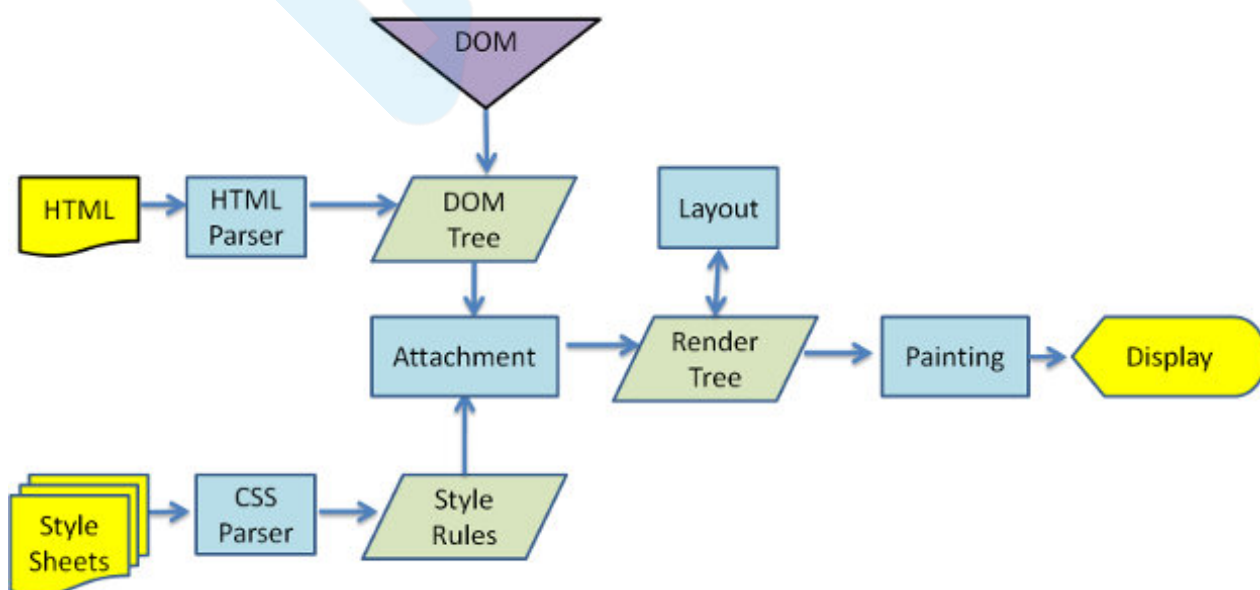
从输入 URL 到页面加载完成，发生了什么？

之前给大家介绍过这个万能面试题，其实这个题的答案，包含着整个互联网运行的过程，我们站在前端的角度，来尝试回答一下这个问题，我们的性能优化策略，也是基于这个面试题的答案，针对每一步依次解析

1. 用户输入kaikeba.com
2. 浏览器通过DNS，把url解析为IP
3. 和IP地址建立TCP链接 发送HTTP请求
4. 服务器接收请求，查库，读文件等，拼接好返回的HTTP响应
5. 浏览器收到首屏html，开始渲染
6. 解析html为dom
7. 解析css 为css-tree
8. dom+ css 生成render-tree 绘图
9. 加载script的js文件
10. 执行js

所谓性能优化，就是上面的步骤加一起，时间尽可能的短，所以基本也有两大方向

1. 少加载文件
2. 少执行代码



DNS

1. 查看dns缓存
2. 本地没缓存，发起dns请求，向本地配置的DNS服务器发请求(递归)

优化： prefetch 预获取，比如使用了cdn的域名

```
2 <meta name="keyword" content="淘宝, 淘宝, 网上购物, C2C, 在线交易, 交易市场, 网上交易, 交易市场, 网上买, 网上卖, 购物网站, 团购, 网上贸易, 安全购物, 电子商务, 放心3
道, 店铺" />
3 <link rel="dns-prefetch" href="//g.alicdn.com" />
4 <link rel="dns-prefetch" href="//img.alicdn.com" />
5 <link rel="dns-prefetch" href="//tce.alicdn.com" />
6 <link rel="dns-prefetch" href="//gm.mmstat.com" />
7 <link rel="dns-prefetch" href="//tce.taobao.com" />
8 <link rel="dns-prefetch" href="//log.mmstat.com" />
9 <link rel="dns-prefetch" href="//tui.taobao.com" />
10 <link rel="dns-prefetch" href="//ald.taobao.com" />
11 <link rel="dns-prefetch" href="//gw.alicdn.com" />
12 <link rel="dns-prefetch" href="//atanx.alicdn.com" />
13 <link rel="dns-prefetch" href="//dfhs.tanx.com" />
14 <link rel="dns-prefetch" href="//ecpm.tanx.com" />
15 <link rel="dns-prefetch" href="//res.mmstat.com" />
16 <link href="//img.alicdn.com/tps/i3/T1OjaVf14dXXa.J0ZB-114-114.png" rel="apple-touch-icon-precomposed" />
17 <style>
18 blockquote, body, button, dd, dl, dt, fieldset, form, h1, h2, h3, h4, h5, h6, hr, input, legend, li, ol, p, pre, td, textarea, th, ul {margin: 0; padding: 0} b
tahoma, arial, 'Hiraqino Sans GB', '\5b8b\4f53', sans-serif {h1, h2, h3, h4, h5, h6 {font-size: 100%} address, cite, dfn, em, var {font-style: normal} c
```

TCP

IP TCP HTTP的关系

1. IP负责找到
2. TCP 负责数据完整性和有序型， 三次握手， 粘包， 滑动窗口等机制
 1. Vs udp
3. http应用层， 负责应用层数据， 数据终止时机

优化策略：

1. 长连接
2. 减少文件体积
 1. js打包压缩
 2. 图片压缩
 3. gzip
3. 减少文件请求次数
 1. 雪碧图
 2. js, css打包
 3. 缓存控制
 4. 懒加载
4. 减少用户和服务器的距离
 1. cdn
5. 本地存储

三次握手

1. 你不在
2. 我在呢
3. 那我开始发数据了呦

文件打包 可以节省这部分优化

HTTP

1. 携带无用的数据，比如header(cookie)
2. 合理利用缓存

浏览器缓存机制

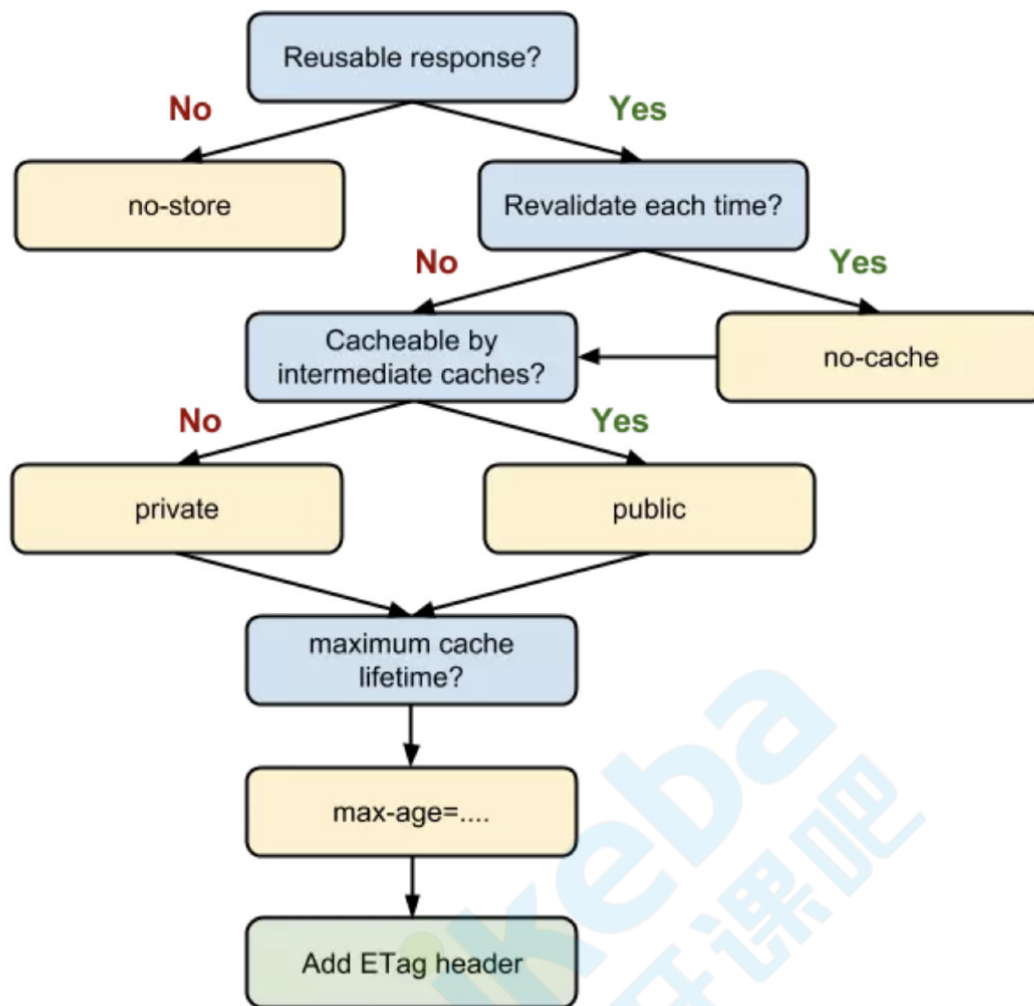
通过网络获取内容既速度缓慢又开销巨大。较大的响应需要在客户端与服务器之间进行多次往返通信，这会延迟浏览器获得和处理内容的时间，还会增加访问者的流量费用。因此，缓存并重复利用之前获取的资源的能力成为性能优化的一个关键方面。

广义的缓存，可以分为这四个，大家对httpcache应该都比较熟悉了

1. Http Cache
2. Service Worker Cache
3. Memory Cache
4. Push Cache

| | | | | |
|-----|------|-------|----------------------|---------|
| | webp | ?... | (from memory cache) | 0 ms |
| | gif | V... | 134 B | 137 ms |
| | gif | V... | 265 B | 140 ms |
| ed) | | V... | 265 B 0 B | 5.00 s |
| | gif | V... | 157 B | 134 ms |
| | gif | V... | 134 B | 135 ms |
| | gif | ta... | 134 B | 11.80 s |
| | jpeg | ta... | (from ServiceWorker) | 6 ms |
| | png | ta... | (from ServiceWorker) | 5 ms |
| | jpeg | ta... | (from ServiceWorker) | 11 ms |
| | jpeg | ta... | (from ServiceWorker) | 10 ms |
| | jpeg | ?... | 49 B | 17 ms |
| | png | ?... | (from memory cache) | 0 ms |
| | gif | V... | 157 B | 137 ms |
| | webp | ?... | (from memory cache) | 0 ms |

http Cache



浏览器大佬：需要获取main.js，看下强缓存里有没有

1. Expires 和 Cache-Control两个header来控制强缓存

```
expires: Wed, 11 Mar 2019 16:12:18 GMT
cache-control: max-age=31536000 // 1.1 精准 优先级高
```

3. 如果命中强缓存，就不会和服务器交互了，直接用缓存

00x400q90.jpg.webp

Request Method: GET

Status Code: 🟢 200 (from memory cache)

Referrer Policy: no-referrer-when-downgrade

如果强缓存失效了，需要执行协商缓存

- 1: 服务器小老弟。浏览器大佬需要main.js 这个文件上次修改

```
If-Modified-Since: Fri, 27 Oct 2017 06:35:57 GMT
```

开课吧web全栈架构师

2. 服务器：小老弟，没改过，直接用缓存把，这次请求返回304 not Modified

如果有etag 类似文件的指纹，这个优先级更高 因为更准确

```
ETag: W/"2aaa-129892f459"  
If-None-Match: W/"2aaa-129892f459"
```

memory Cache

内存缓存，短命 比如常用数据存js里，浏览器也有自己的策略，base64图片，体积小的静态资源

Service Worker Cache

Service Worker 是一种独立于主线程之外的 Javascript 线程。它脱离于浏览器窗体，算是幕后工作，可以实现离线缓存，网络代理等

```
window.navigator.serviceWorker.register('/kaikeba.js').then(  
  function () {  
    console.log('注册成功')  
  }).catch(err => {  
    console.error("注册失败")  
  })
```

push cache

http2的缓存

文件打包

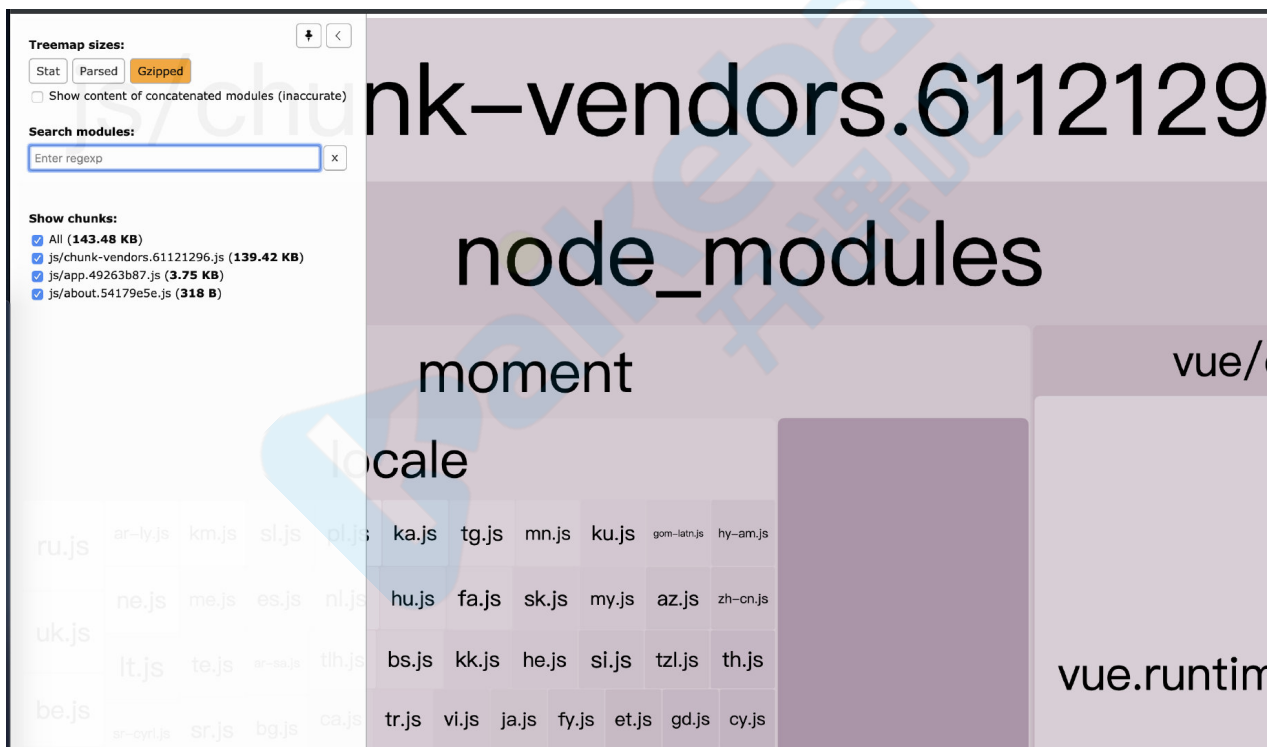
分析文件大小

```
npm install lodash echarts moment -D
```

```
const BundleAnalyzerPlugin = require('webpack-bundle-analyzer').BundleAnalyzerPlugin;

module.exports = {
  configureWebpack: {
    plugins: [
      new BundleAnalyzerPlugin()
    ]
  }
}
```

```
import moment from 'moment'
import _ from 'lodash'
Vue.config.productionTip = false
console.log(moment())
console.log(_.max([5,4,1,6,8]))
```



或者执行vue ui

serve
编译和热更新 (用于开发环...)

build
运行中

lint
检查并修复源文件

inspect
已完成

build 编译并压缩 (用于生产环境)

vue-cli-service build

■ 停止 ⚙ 变量 📄

📄 输出 🛠 控制台 🔍 分析

🛠 控制台 Parsed ⓘ

状态
编译成功

错误
0

警告
2

资源
442.7kB (Parsed)

模块
429.9kB (Parsed)

依赖项
425.2kB 98.92%

Idle (5s)

速度统计

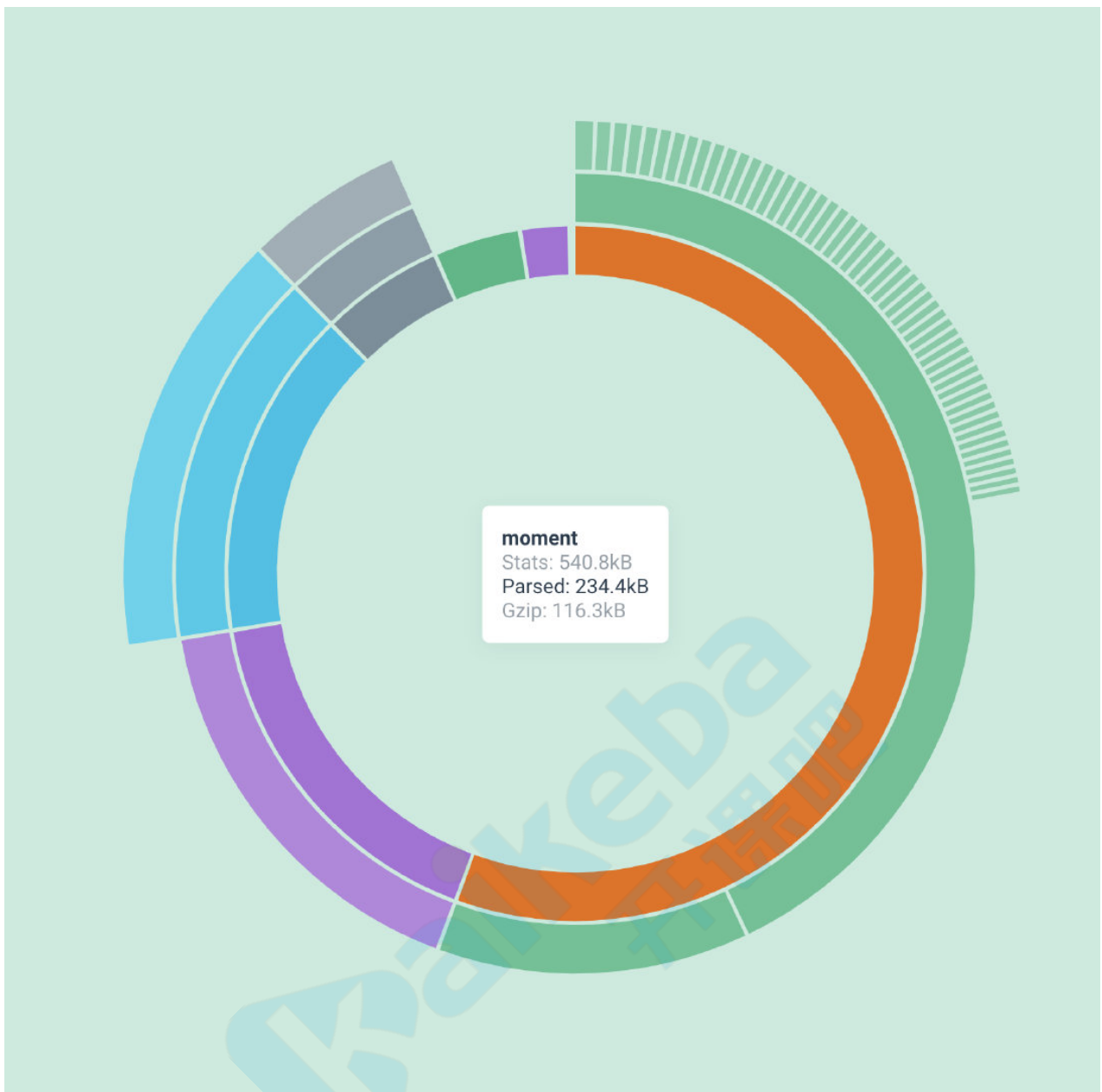
| | | | | | | | |
|----------------|--------|-------------|--------|-------|--------|---------|-------|
| Global Average | 0.52s | Mobile Edge | 15.25s | 2G | 13.15s | 3G Slow | 9.05s |
| 3G Basic | 2.46s | 3G Fast | 2.31s | 4G | 0.55s | LTE | 0.36s |
| Dial Up | 69.29s | DSL | 2.36s | Cable | 0.72s | FIOS | 0.18s |

资源

| | Parsed | Global | 3G Slow | 3G Fast |
|----------------------------------|---------|--------|---------|---------|
| js/chunk-vendors.61121296.js | 422.9kB | 0.5s | 8.66s | 2.21s |
| js/app.49263b87.js | 10.4kB | 0.04s | 0.6s | 0.2s |
| img/logo.82b9c7a5.png | 6.7kB | 0.04s | 0.53s | 0.18s |
| favicon.ico | 1.1kB | 0.03s | 0.42s | 0.16s |
| index.html | 0.7kB | 0.03s | 0.41s | 0.15s |
| js/about.54179e5e.js | 0.4kB | 0.03s | 0.41s | 0.15s |
| css/app.3c0b035c.css | 0.4kB | 0.03s | 0.41s | 0.15s |
| js/chunk-vendors.61121296.js.map | 2.0MB | 2.26s | 39.42s | 9.9s |
| js/app.49263b87.js.map | 42.4kB | 0.08s | 1.23s | 0.36s |
| js/about.54179e5e.js.map | 1.4kB | 0.03s | 0.43s | 0.16s |

依赖项

| | |
|------------|---------|
| moment | 238.7kB |
| lodash | 70.7kB |
| vue | 64.2kB |
| vue-router | 24.1kB |
| core-js | 17.0kB |
| vuex | 9.6kB |
| vue-loader | 0.8kB |



如果我们改成只引入lodash需要的模块，moment换成更小的dayjs

打包后的大小从464kb (gzip之后143kb) 下降成

| File | Size | Gzipped |
|---|------------|------------|
| dist/js/chunk-vendors.61121296.js | 422.87 KiB | 139.76 KiB |
| dist/js/app.49263b87.js | 10.43 KiB | 3.75 KiB |
| dist/js/about.54179e5e.js | 0.44 KiB | 0.31 KiB |
| dist/css/app.3c0b035c.css | 0.42 KiB | 0.26 KiB |
| Images and other types of assets omitted. | | |
| DONE Build completed. The dist directory is ready to be deployed. | | |

Use Ctrl+C to close it

| File | Size | Gzipped |
|-----------------------------------|------------|-----------|
| dist/js/chunk-vendors.c54ad645.js | 124.32 KiB | 43.56 KiB |
| dist/js/app.2fb15299.js | 6.08 KiB | 2.30 KiB |
| dist/js/about.54179e5e.js | 0.44 KiB | 0.31 KiB |
| dist/css/app.3c0b035c.css | 0.42 KiB | 0.26 KiB |

Images and other types of assets omitted.

删除冗余代码的tree-shaking，和去除无效代码，我们webapck那里都介绍过了 这里不赘述了

如果是个别页面使用了echarts这种库，一定记得懒加载


图片优化

图片通常是最占用流量的，PC端加载的平均图片大小时600K，简直比js打包后的文件还大了，所以针对图片的优化，也是收益不错的

不同的场景，使用不同的文件里类型

1. jpg
 1. 有损压缩
 2. 体积小 不支持透明
 3. 用于背景图，轮播图
2. png
 1. 无损压缩，质量高，支持透明
 2. 色彩线条更丰富，小图，比如logo，商品icon
3. svg
 1. 文本，体积小 矢量图
 2. 渲染成本，学习成本

图片打包雪碧图 减少http请求次数 webpack-spritesmith

| Name | × | Headers | Preview | Response | Timing |
|---|---|---------|---------|---|--------|
| <input type="checkbox"/> s.gif | | | | | |
| <input checked="" type="checkbox"/> TB1UDHOcwoQM... | | | |  | |
| <input type="checkbox"/> TB1BlobNFXXXXX... | | | | | |
| <input type="checkbox"/> TB1tyFSXm_l8KJj... | | | | | |
| <input checked="" type="checkbox"/> TB1Z_HcQFXXXXX... | | | | | |
| <input type="checkbox"/> data:image/webp;... | | | | | |
| <input type="checkbox"/> v.gif?logtype=1&tit... | | | | | |

gzip

accept-encoding: gzip 开启gzip

HTTP 压缩就是以缩小体积为目的，对 HTTP 内容进行重新编码的过程

Gzip 压缩背后的原理，是在一个文本文件中找出一些重复出现的字符串、临时替换它们，从而使整个文件变小。根据这个原理，文件中代码的重复率越高，那么压缩的效率就越高，使用 Gzip 的收益也就越大。反之亦然。

基本上来说，Gzip都是服务器干的活，比如Nginx

本地 存储

cookie localstroage, sessionStroage, indexDB

1. cookie
 1. 最早，体积先定，性能浪费，所有请求都带上所有当前域名的cookie
2. Web Storage
 1. Local Storage 与 Session Storage
 2. 存储量大，不自动发给服务端，js控制
3. indexdb
 1. 运行在浏览器上的非关系型数据库
4. PWA
 1. 基于缓存技术的应用模型

CDN

海南的哥们，访问开课吧，光电线就要那么远，肯定慢，所以我们可以吧静态资源，部署在分布式的cdn上，海南的哥们，就近获取资源，比如广州机房，

cdn单独的域名，浏览器并发获取

服务端渲染

如果是SPA 首屏SSR就是性能优化的重要一环

nuxt 和 next

vue服务端渲染

```
const Vue = require('vue')
// 创建一个express应用
const server = require('express')()
// 提取出renderer实例
const renderer = require('vue-server-renderer').createRenderer()

server.get('*', (req, res) => {
  // 编写Vue实例（虚拟DOM节点）
  const app = new Vue({
    data: {
      url: req.url
    },
    // 编写模板HTML的内容
    template: `<div>访问的 URL 是: {{ url }}</div>`
  })

  // renderToString 是把Vue实例转化为真实DOM的关键方法
  renderer.renderToString(app, (err, html) => {
    if (err) {
      res.status(500).end('Internal Server Error')
      return
    }
    // 把渲染出来的真实DOM字符串插入HTML模板中
    res.end(`
      <!DOCTYPE html>
      <html lang="en">
        <head><title>Hello</title></head>
        <body>${html}</body>
      </html>
    `)
  })
})

server.listen(8080)
```

nuxt.js 服务端渲染框架体验

1. 基于Vuejs
2. 服务端渲染
3. 路由
4. 热加载
5. 支持http2

react服务端渲染

```
import express from 'express'
import React from 'react'
import { renderToString } from 'react-dom/server'
import App from './App'

const app = express()
// renderToString 是把虚拟DOM转化为真实DOM的关键方法
const RDom = renderToString(<App />)
// 编写HTML模板，插入转化后的真实DOM内容
const Page = `
  <html>
    <head>
      <title>test</title>
    </head>
    <body>
      <span>ssr </span>
      ${RDom}
    </body>
  </html>
`

app.get('/index', function(req, res) {
  res.send(Page)
})

// 配置端口号
const server = app.listen(8000)
```