目标

输入一套可优化性能的可复用可参考的方案。

计划

1. 分析经过几个月优化的页面。
2. 站在巨人肩膀上明确优化方向。
3. 具体可执行优化的方案。
4. 未来还有哪些可以去做的事情。

详细

1. 分析经过几个月伙伴们优化的页面。

他们的共同优化思路特点

1. 静态资源做了缓存。
2. 各个包的大小不超过 50kb
3. index.html初始化中的代码极少。
4. 减少各种文件的体积。

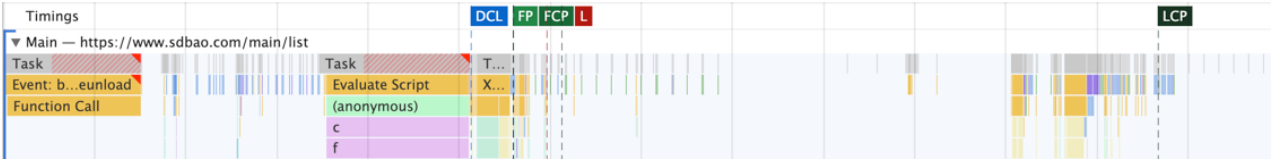
结论：

复杂的页面，评分达到60~70，TTI（可交互时间）降到5s~6s以下，较为优秀

参考链接： https://wiki.shuiditech.com/pages/viewpage.action?pageId=613517955

2. 站在巨人肩膀上明确优化方向。

水滴保险 https://www.sdbao.com/main/list

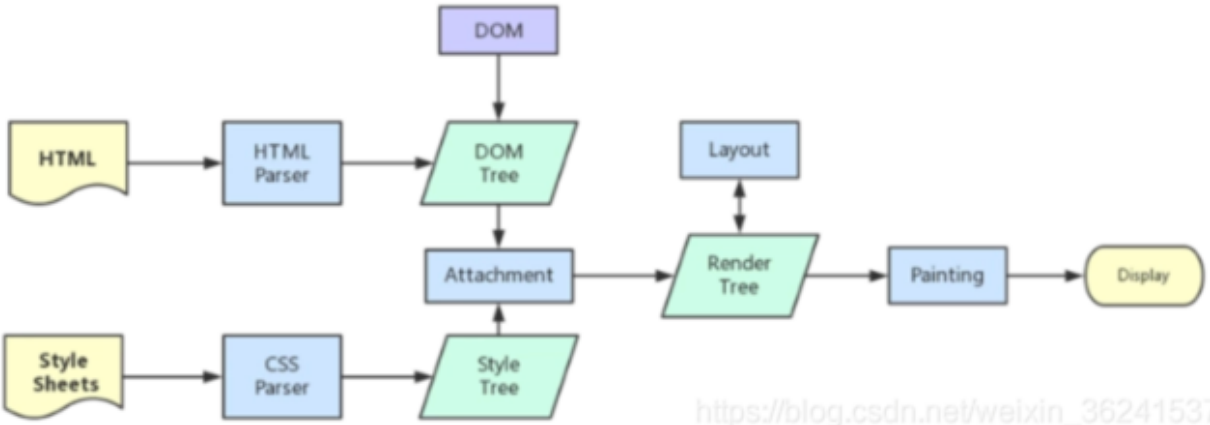


页面优化要抓重点，站在‘巨人’的肩膀上，这个巨人就是之前性能优化小组总结的一些经验。

1. task不宜过长，否则会影响加载性能。
2. js文件拆分50kb得到的加载反馈最优。
3. 对性能最差页面，单独使用ssr去优化。

站在肩膀上远望

页面渲染我们目前明确的样子如下：

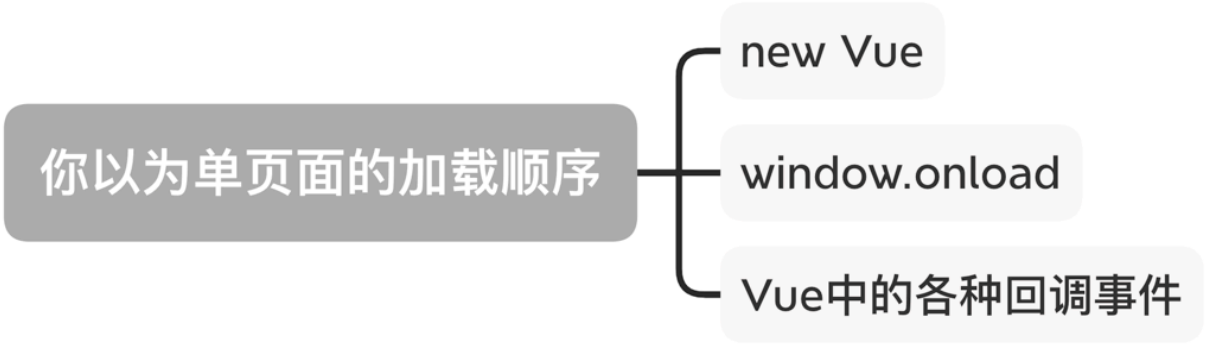


这里可以得出

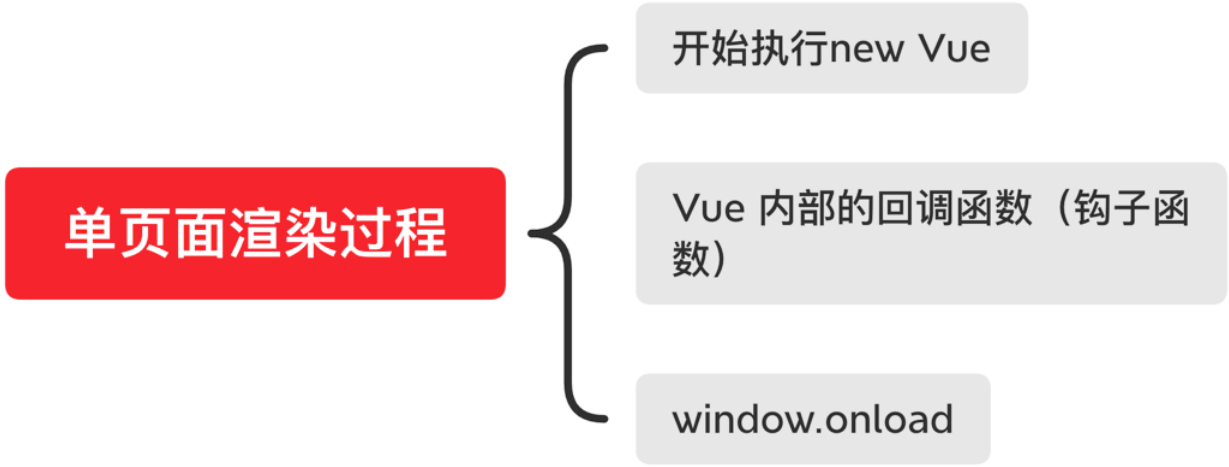
1. 首先解析HTML文件，构建DOM树。解析CSS，构建styleTree。当二者解析完成后，合二为一构建renderTree，最终浏览器绘制并呈现。
2. css的加载并不会阻塞dom树的构建，但是需要等styleTree构建完之后，才能创建渲染树，因此css加载会阻塞renderTree的构（加载js资源时，会暂停html的解析，因此会阻塞页面加载）

> 而当中第二个结论我们可以得出加载js文件统一会影响解析。因此很多小伙伴喜欢把 - **js文件放在html底部**。
>
> 但是我们目前抓取用户可使用时长不仅仅到这，放在底部对于性能的影响其实对目前的首屏影响不是特别大，仅仅是大家目前都可以这么做。

单页面中的你以为执行过程



真实的事件执行顺序



---

lighthouse参数和要求

First Contentful Paint（FCP） 浏览器首次渲染任何内容开始时间。

Time to Interactive 可交互时间。

Speed Index代表页面内容渲染所消耗的时间。

TBT（Total Blocking Time）衡量从FCP到TTI之间主线程被阻塞时长的总和。规定持续时间超过50ms的任务为长任务，低于50ms的任务为短任务。长任务中超过50ms的时间被认为是“阻塞”的，因此，TBT是所有长任务中阻塞时间的总和。

Largest Contentful Paint (LCP) 用于衡量标准报告视口内可见的最大内容元素的渲染时间。LCP 应该在页面首次开始加载后的 2.5 秒内发生。

Cumulative Layout Shift (CLS) :衡量视觉稳定性，为了提供良好的用户体验，页面的CLS应保持小于0.1。

Performance节点解释

1. DOMContentLoaded（DCL） 当初始的 HTML 文档被完全加载和解析完成之后,DOMContentLoaded 事件被触发,而无需等待样式表、图像和子框架的完成加载。

2. First Paint(FP)**，是Paint Timing API的一部分，是页面导航与浏览器将该网页的第一个像素渲染到屏幕上所用的中间时，渲染是任何与输入网页导航前的屏幕上的内容不同的内容。它回答了“发生了什么？”这个问题。

3. First Contentful Paint（FCP） 浏览器首次渲染任何内容开始时间。

> 衡量页面开始加载到页面中第一个元素被渲染之间的时间。元素包含文本、图片、canvas等。

4. load （L） : window.onload执行节点

5. Largest Contentful Paint (LCP) 用于衡量标准报告视口内可见的最大内容元素的渲染时间。为了提供良好的用户体验，网站应努力在开始加载页面的前 2.5 秒内进行 最大内容渲染 。

> LCP 考虑哪些元素
>
> LCP 目前并不会计算所有元素，因为这样会使这个指标变得非常复杂，它现在只关注下面的元素：
>
> - 元素
>
> - 元素内的 元素
> -
>
> 元素
> - 通过 url 函数加载背景图片的元素
> - 包含文本节点或其他内联文本元素子级的块级元素。
>
> 为了在开始时保持简单，将元素限制到这个有限的集合是有意的。随着研究的深入，将来可能会添加更多的元素。

其他参考

First Input Delay (FID) : 衡量可交互性，为了提供良好的用户体验，页面的 FID 应当小于 100毫秒。

Cumulative Layout Shift (CLS) :衡量视觉稳定性，为了提供良好的用户体验，页面的CLS应保持小于 0.1。

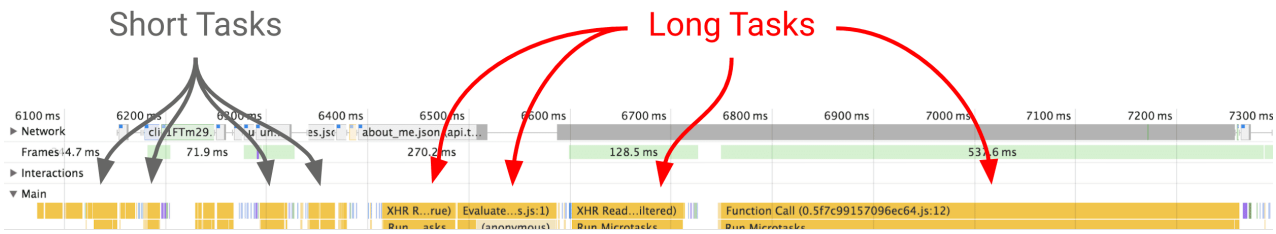
---

3. 具体可执行优化的方案。

代价最小立竿见影最快的

1. 避免long task。

> 由于单页面组件在渲染页面时候很多情况下需要请求服务器的数据，并且按照前端的template去处理这些数据，在首屏加载过程中**如果你需要写各种循环或者判断才成把后端的数据转换成你想要的数据，毫无疑问这将严重的影响首屏的加载时间。**

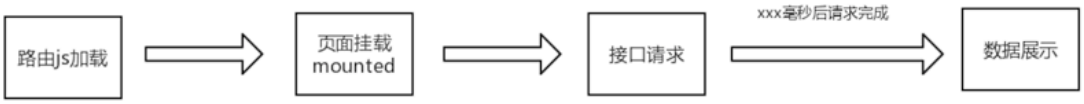


> Are long JavaScript tasks delaying your Time to Interactive
>
> 减少长期任务占用主线程的时间

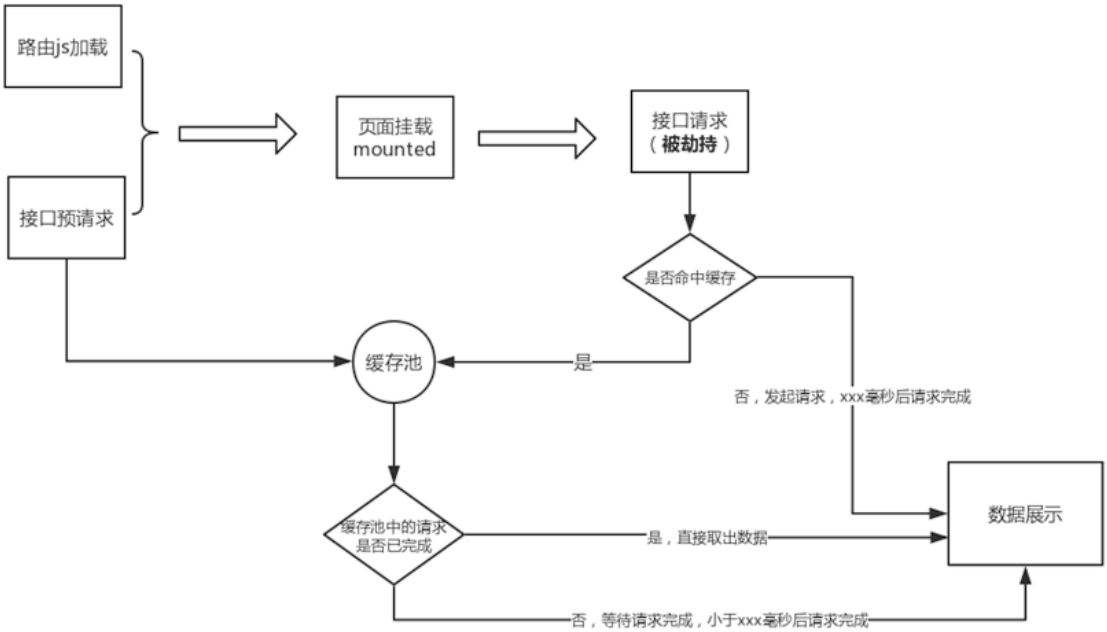
- 和后端沟通，首屏展示数据必须要和前端template中的模版对齐，把这类的数据处理放到后端。
- 减少在mounted这类回调函数中的其他函数执行，对于非要执行的js函数建议在 x6 cpu情况下计算下耗时如果超过 60ms 需要优化，因为我们客户的设备远远达不到 slow x6 cup ，这类的设备执行时间可能接近 500s 如果多几个这样的函数肯定影响首屏。

2. 资源文件的拆分不要有超过 50kb的静态资源文件，否则首次加载白屏会很久，直接影响用户体验。
3. 已知大小的图片尽量加上宽高。
4. 不要过分的依赖接口预先请求因为其流程如下：

普通的单页面应用执行过程



经过该方案优化后的执行过程



4. 未来可期

1. 预渲染配合webpack使用 prerender-spa-plugin,缺点局限性很大，适合静态宣传页。
2. 客户端预渲染。可能要改写vue生命周期风险较大。
3. 服务端预渲染。可能影响业务风险较大。

总结

目前 3类似于西药-特效药，可以很快的帮我们解决一部分难题，但想要图片需要从4下手。