# 一.网站主要性能指标列表及其说明

- [Navigation Timing](#navigation-timing)

- Navigator Interface

- [Resource Timing](#resource-timing)

- [Element Timing](#element-timing)

- Service Worker Status

- StorageManager interface

- Time to First Byte ([TTFB](https://web.dev/ttfb/#what-is-a-good-ttfb-score))

- First Contentful Paint ([FCP](https://web.dev/first-contentful-paint/))

- Largest Contentful Paint ([LCP](https://web.dev/lcp/))

- First Input Delay ([FID](https://web.dev/fid/))

- Cumulative Layout Shift ([CLS](https://web.dev/cls/))

- Interaction to Next Paint ([INP](https://web.dev/inp/))

- Total Blocking Time ([TBT](https://web.dev/tbt/))

- Navigation Total Blocking Time ([NTBT](#navigation-total-blocking-time-ntbt))

重要指标简介：

1. FCP: 首个内容绘制时间，浏览器开始渲染内容所花费的时间。FCP指标将显著影响用户对性能的感知。在FCP之前，用户只能看到一个**空白的屏幕**。
2. LCP: 最大内容绘制时间，发生在浏览器的视窗完全加载时。视区之外的其他内容可能仍在加载中，但LCP是当用户感到页面是可见时。【服务端渲染可显著提高LCP评级】
3. TTI: 首次可交互时间，用户用鼠标、键盘能够与页面进行交互之前所花费的时间。在Vue中，它发生在完成第一次渲染之后，当Vue附加了事件处理程序时。
4. TBT: 总阻塞时间，是FCP和TTI之间发生的**所有阻塞任务的**总时长。**阻塞任务**是任何超过50ms的任务，肉眼所见会让人感觉卡住了。TBT通常通过运行更少的js代码或减少DOM的操作次数而得到改进，**代码分割**可能很有效。
5. CLS: 累积布局偏移，是衡量网页的跳跃性或视觉稳定性的指标。如果应用程序在页面加载期间插入的页面元素影响到其他内容的位置，就会影响CLS指标。CLS是加载过程中移动的页面的比例。

# 使用浏览器性能工具

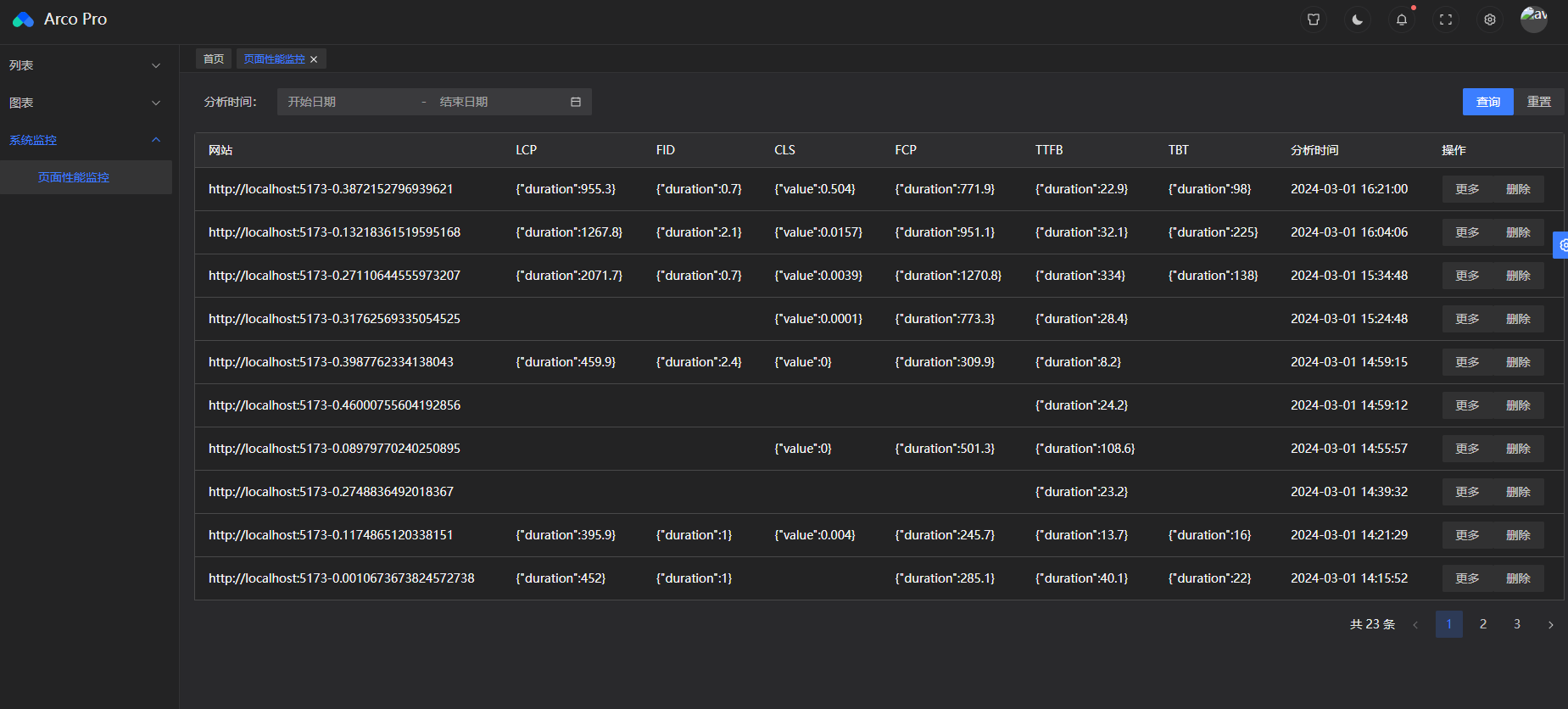
1. 使用Lighthouse
2. 浏览器内置的Performance选项卡

# 三、性能指标数据上报、查看、分析

1.目前存储需要后端服务支持上报数据接口：@/api/frontStorage.js

2.查看、分析

2.1 查看网站首屏加载的基础指标，按照重要性目前开放表格中以下几项：



分析标准：

| Web Vitals | Good | Needs Improvement | Poor |

| ------------------------------------- | ----------: | ---------------------------: | -------------: |

| Time to First Byte (TTFB) | 0-800 | 801-1800 | Over 1800 |

| Redirect Time (RT) | 0-100 | 101-200 | Over 200 |

| First Contentful Paint (FCP) | 0-2000 | 2001-4000 | Over 4000 |

| Largest Contentful Paint (LCP) | 0-2500 | 2501-4000 | Over 4000 |

| First Input Delay (FID) | 0-100 | 101-300 | Over 300 |

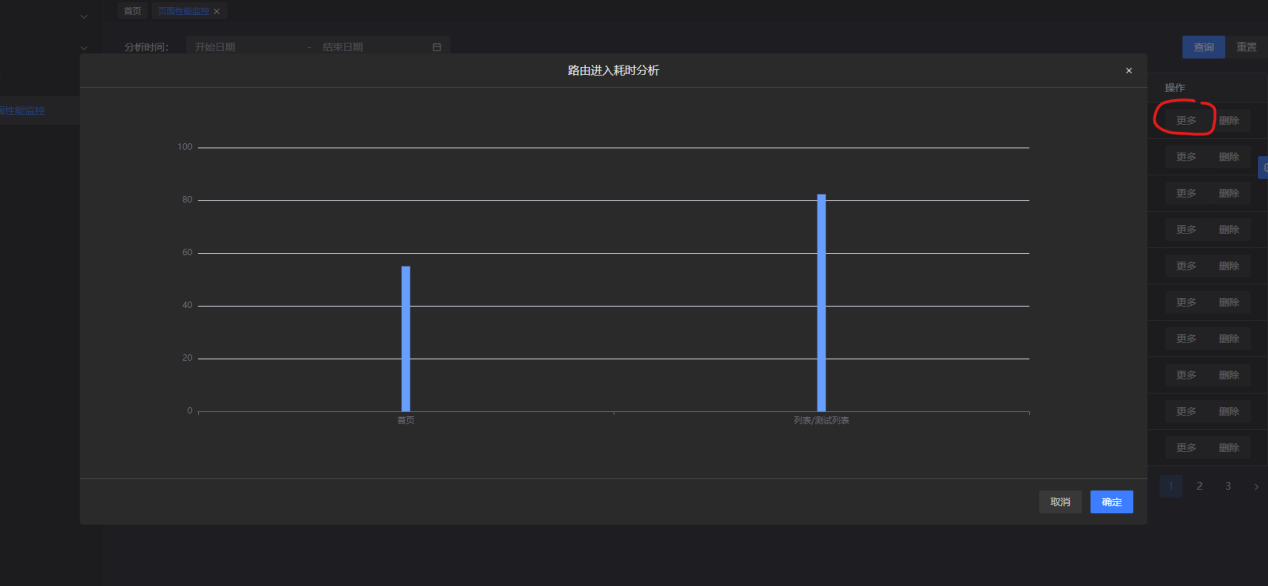
| Cumulative Layout Shift (CLS) | 0-0.1 | 0.11-0.25 | Over 0.25 |

| Interaction to Next Paint (INP) | 0-200 | 201-500 | Over 500 |

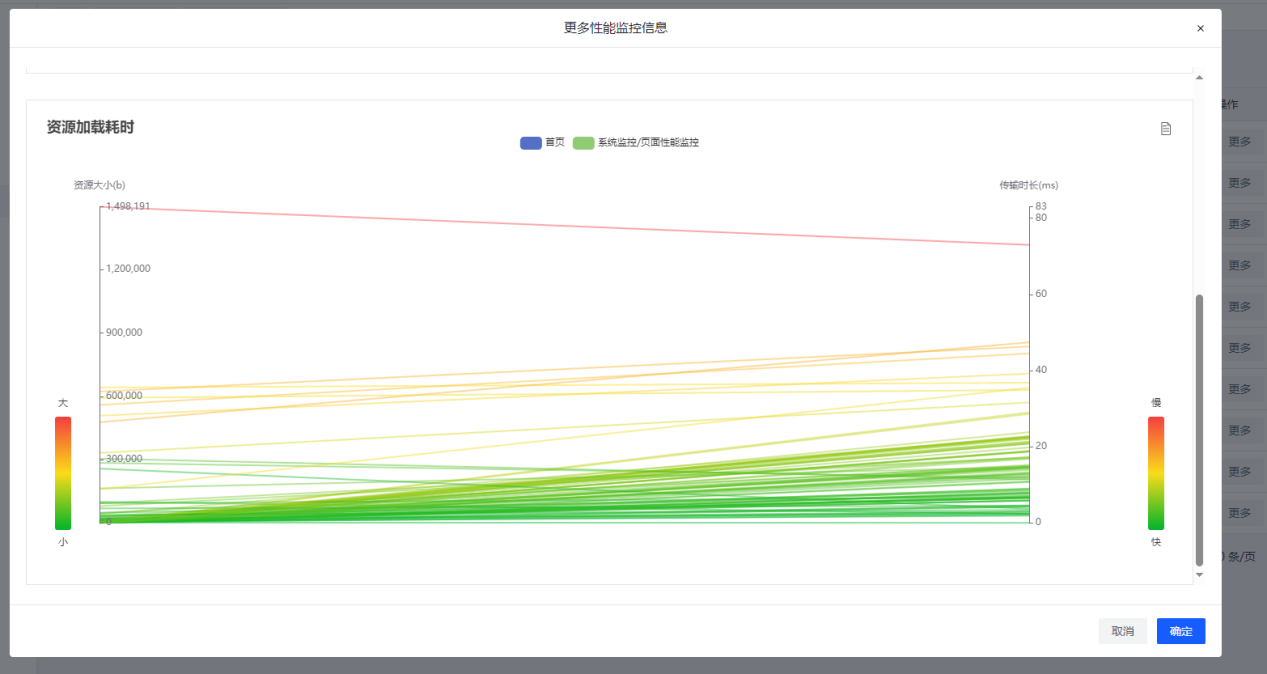
| Total Blocking Time (TBT) | 0-200 | 201-600 | Over 600 |

| Navigation Total Blocking Time (NTBT) | 0-200 | 201-600 | Over 600 |

2.2 对于SPA网页应用，路由组件是通过异步加载的，“路由进入耗时分析”柱状图表可查看其“导航触发”-“导航被确认”的时长（ms）,导航解析渲染流程请参考： [导航守卫](https://router.vuejs.org/zh/guide/advanced/navigation-guards.html)



2.3 网站的活动过程中，会加载各种静态资源：js,css,image(jpeg, png, svg, ...),model(glb,...),...等等。后台脚本（@/utils/perfumeAnalytics.js）会根据浏览器PerformanceObserver API上报资源的性能数据，目前从中提取大小（encodedBodySize）、传输时长（duration）这2项指标，绘制到图表中。粗略地以资源上报时所属的页面路由进行groupBy,如图：



# 三.对网站进行性能优化

建议“当你能够精确地测量性能时，才考虑提高代码的速度”，采用优化的代价是：a开发成本大 b 容易引入新的bug或其他维护问题

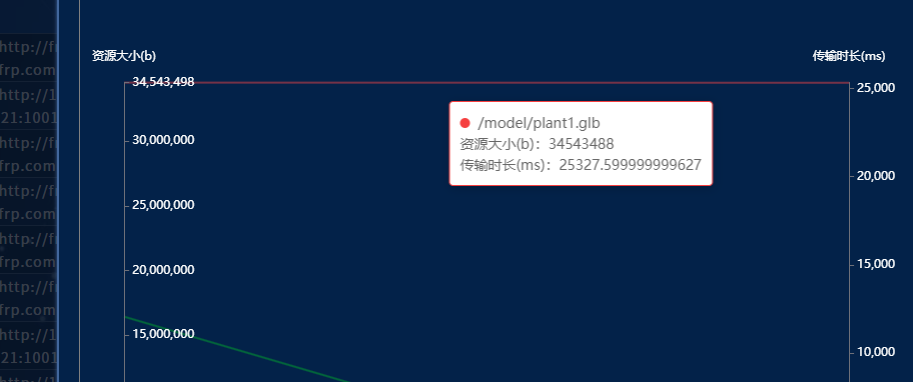
性能指标数据量通常随着项目增长而膨胀，关注针对图表中的top6数据通常会取得更好的优化效益：

1. 首页：

1.1尽量只加载首屏用到的功能模块，在用到的时候使用“按需加载”加载相应的资源，如路由异步加载、代码分割；

1.2 考虑使用SSR技术（需要权衡服务器开销、白屏、SEO、服务端渲染开发等性价比）；

1. 资源大小/时长监控：



建议手段：缓存（如nginx配置）、压缩（gzip, Brotli,...）、CDN、域名分片、dns-prefetch、懒加载、预加载、...

1. 图像：压缩（tinyPng、webp）、切片（针对大型图像）、精灵图、视窗懒加载、脚本绘制代替图片动画、...
2. 字体：UI提供的排版效果丰富的字体较大，使用工具压缩：如font-spider
3. 减小源码的体积（大小）：

5.1常见的情况是重复冗余的代码块，抽象成函数或者组件

5.2 Tree-shaking：尽量使用ES模块格式的依赖包，如：lodash-es比lodash更好

5.3 各种依赖包的引入会导致代码体积膨胀，评估用到的功能与体积之间的性价比

6. 更多......使用动画等，待补充。

# 四、参考

性能优化涉及的范围比较广,深度优化比较耗成本，不能立杆见影。

1.[前端性能精进之优化方法论（一）——测量 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/607781009?utm_id=0)

2.[性能优化 | Vue.js (vuejs.org)](https://cn.vuejs.org/guide/best-practices/performance.html)

3.[evaluating-page-experience](https://webmasters.googleblog.com/2020/05/evaluating-page-experience.html)

4.[web-vitals](https://github.com/GoogleChrome/web-vitals)

5.[Perfomance audits](https://www.youtube.com/watch?v=zDcbpFimUc8)

6. [web.dev 指南](https://web.dev/fast/" \t "https://cn.vuejs.org/guide/best-practices/_blank)