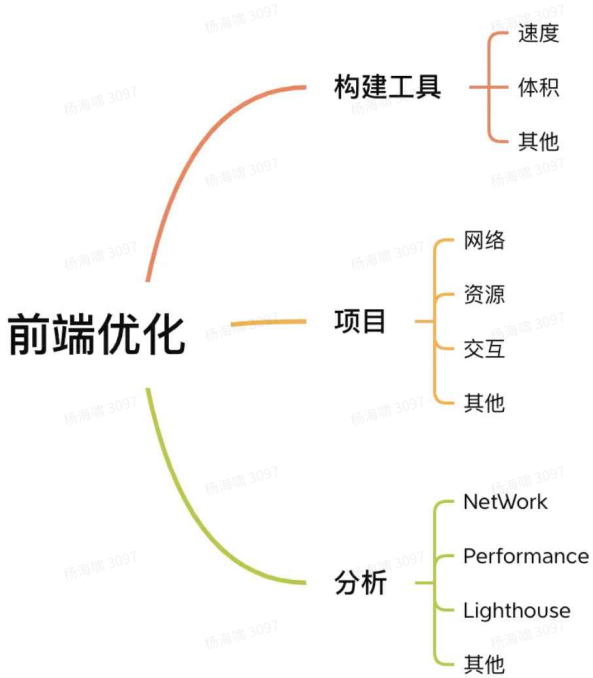
浅谈前端项目性能优化实践

如何下手?

首先，我们可以先对性能优化的方向和一些常见的性能指标有个大致了解。可以读一下

- 采集性能指标标准
- 雅虎团队分享的黄金守则

在了解了这些后，我们再从一个简单的导图去看，后面的优化就可以由点到面的去进行



那么，从这个导图出发，整体的优化逻辑：

分析

从分析出手，任何需要做的事情在没有明确具体需求的过程中，都是很难去行动的。举个例子：

> 某天，某销售被安排了一个很重要的客户去接待，说让他用尽一切办法一定要把客户签下来。

接到任务的时候，大多数人都是一头雾水，或许有经验的话以前做过相关的工作，能有一些通用手段可以勉强应付，但是人与人之间是不同的，所以要让客户满意，必须得下一番功夫对客户做一些调研，方便下一步投其所好的安排接待工作。

故事结束。🛎️

回到事情本身，前端项目优化也是一样的道理。接到任务的时候，大部分同学应该都知道一些通用的优化手段，但是想要再进一步去实现这些手段，却总感到无从下手。因此还是需要和接待客户一样，先分析当前的项目，发现问题，针对问题的本身分析后才能去实施具体的优化手段。

分析工具

怎么分析项目呢？如果我们入手一个项目，仅知道慢，是无处下手的，如何发现导致慢或卡顿的原因?

项目分析

项目分析落实到了具体的点上，针对当前项目的具体现状来去做问题task的点对点优化。而分析项目的手段大体上有以下几个。

- **NetWork:** 网络面板，用于侦测浏览器资源下载与上传的能耗视图和观测资源的瀑布流
- **Performance(Performance insights):** 性能面板：用于侦测浏览器运行时的性能表现，得出项目运行的时序图，同时也可以分析页面的一些隐式问题，如 (内存泄漏)。
- **Lighthouse:** 性能评测(灯塔)，基于当前浏览器页面的阶段性加载现状来为用户提供结果分析数据指标报告。
- **性能监控:** 如目前公司内部Web项目接入的腾讯rum监控

构建工具分析

针对构建工具的优化，无非就是两个方面：

- 体积：包体积是否存在杂质，能不能过滤出去，或者想办法减小它的影响?
- 构建速度：构建速度影响开发调试效率和发布相关环境的效率。

传统的Web项目往往是基于webpack或者vite去进行构建，而小程序往往是通过微信官方提供的开发者工具去进行构建。我们可以借助这些构建工具提供的代码依赖分析插件，如：`webpack-bundle-analyzer` / `rollup-plugin-visualizer` 等来分析当前项目中是否存在 `重复依赖` ，`影子依赖` ，`模块体积差异` 等等问题，使用交互式可缩放树图 `可视化` 的来梳理自己项目应用中的问题

如何进行优化?

项目角度

1. 定义优化目标：明确优化的目标，如缩短首屏时间或某页面加载时间、提升某功能用户体验等。
2. 评估现状：使用性能分析工具，获取项目的性能数据和关键指标，了解当前项目的性能状况。

常见指标包括：

- FCP（首次内容绘制）：衡量首屏加载时间。
- LCP（最大内容绘制）：衡量页面上可见内容最长绘制时间。

    - CLS（累积布局偏移）：衡量页面在加载过程中的布局变动情况。
    - IDLE（白屏时间）：衡量页面白屏时间。
3. 发现问题：根据收集到的性能数据，再结合代码分析，从资源加载、网络相关和交互逻辑等几个方面入手，梳理项目渲染的流程图，确定导致性能问题的瓶颈，如耗时较长的操作和资源加载问题等。
4. 制定优化方案，结合关键性能指标和性能优化手段，提出优化措施。

下面列举一通用常见的手段

- `资源加载` 方面

懒加载：延迟加载非关键资源，提高页面初次加载速度。这项技术被广泛用于spa应用、图片资源等非阻塞性资源的优化上。对于缩短关键页面渲染非常有帮助，如各位同学耳熟能详的首屏优化。

预加载：预加载的目的就是通过在渲染前预先拿到需要展示的数据，从而缩短关键承流页面呈现给用户的渲染时间，大多数情况下可以分为首页预载和关键页预载两方面。

- 首屏预载：Web端对 `rel` 属性提供了相关的pre-load、pre-render、pre-fetch相关API实现。小程序方面也提供了相关冷启动获取数据的能力，例如 `getBackgroundFetchData` 。
- 关键页预载：关键页预载的话就根据自身业务来实现了，参考商家主页中优惠券详情页、营销活动等页面都是用户比较熟悉的页面。因此，可以考虑页面跳转与数据请求共同进行。

接口优化：耗时较久的接口如果阻塞核心链路，可以将其拆成多个粒度小的接口来渐进式的调用以此来保证页面优先加载。反之亦然，如果接口零散占用了太多的请求资源，则可以将其聚合成一个完整的大接口来获取数据。同时需要注意大部分浏览器限制http1.0同域名接口并发数是在6个，如果超出会以队列的形式依次执行

缓存：浏览器缓存策略大体上分为强缓存和协商缓存。利用好缓存策略可以有效的降低资源的重复加载，从而提高网页的整体加载速度。

- `网络相关`

DNS预解析： 了解DNS解析过程的同学都知道，相当耗时。而浏览器也提供了dns-prefetch的相关操作，用于抢先对origin进行解析。

PreConnect：既然DNS可以预解析，那么请求连接的话也相对应的可以抢先执行。

内容分发网络CDN：通过内容分发网络CDN可以使用户就近获取所需要的内容资源，从而降低网络拥塞，提高用户访问响应速度和保证访问资源的命中率。

Gzip：开启Gzip的压缩方案来减少网络实际传输数据的大小，服务器会将请求资源进行压缩后传输给客户端。

> 优化的手段很多，使用不当可能会带来负提升，不一定都适用。根据项目现状和优化目标选择最适合的手段

5. 逐一实施优化措施，并收集优化效果数据，与优化前进行对比。
6. 测试与监测：

- 使用真实设备测试：确保在真实设备上的性能表现。
- 监测工具：使用性能监测工具实时监测项目的性能，并进行持续优化。

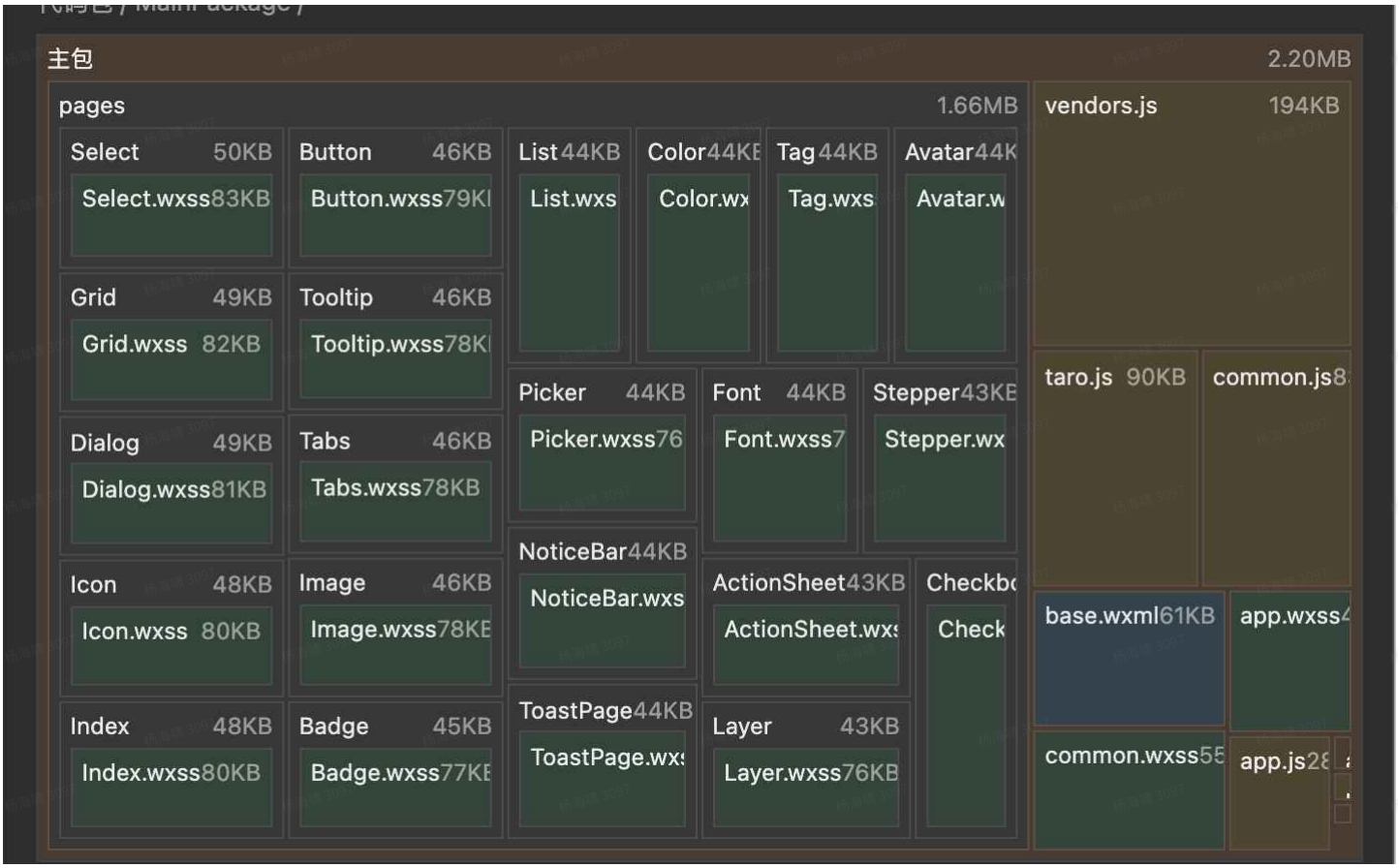
7. 将优化点和效果记录在文档中，方便后续维护和团队参考。

构建工具角度

通过依赖分析插件来进行构建产物的分析。那么分析出结果后？又该怎么去做一些优化呢？

首先我们可以根据得到的分析视图，看一下我们项目的代码依赖是怎么分布的。

举个例子。这是以前小程序组件库发现的一个问题（开源后已经修复了）



> 可以看到每个组件的wxss文件比其js文件还要大得多，比如Select.wxss 83kb而Select.js 50kb。

通过分析后发现，组件的样式文件中引入了全部组件的样式文件，这是不合理的，也是导致文件这么大的原因。通过依赖分析，我们可以很容易分析文件的关系，发现诸如：文件大小不合理、打包了无用依赖、过多的静态资源等等。

同理，我们还可以针对包体积大小做一些调整，能做的事情大体方向有以下几点：

- **按需加载**：针对项目中比较大型的工具类库，如lodash、UI Design、use-hooks等类库，如果只是使用很小的一部分，就需要避免全量加载的情况。
- **tree-shaking**：通过ESM的特性，可以在构建阶段将项目模块中的死代码、无用代码进行移除，从而保证干净的结构和最小的文件大小。
- **minify**：通过webpack相关的插件，可以为代码产物做混淆压缩，如JS minify与CSS minify，当项目的代码越大的时候，压缩的效果就越明显。
- **CDN**：对于部分共有依赖、静态资源可以将其放置在CDN中来进行加载，能够极大幅度的减少本地项目产物的体积大小，缺点就是无法进行按需加载，占用浏览器请求资源，具体的收益可以自己权衡。

栗子

最后，一个栗子 国际化独立站首屏性能分析及优化技术方案

参考阅读

毫秒级突破!腾讯技术团队是如何做前端性能优化的? - 掘金

[转] 雅虎团队:网站性能优化的35条黄金守则_51CTO博客_雅虎性能优化规则