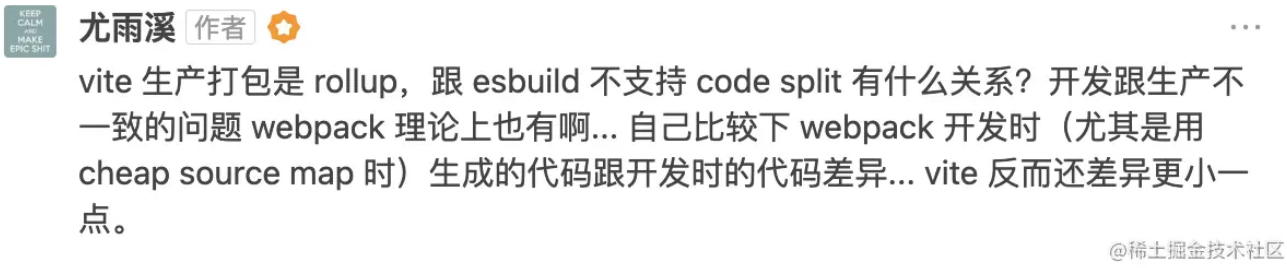
聊聊我与Vite这一年以来的磕磕绊绊

相识

第一次听到你的名字，是在去年（2021）的vueConf，当时我正在被webpack漫长的打包编译过程折磨到抓狂。PPT里介绍的关于你的每一个特性都像是为我当时遇到的痛点量身定制的。

换句肉麻的话，在那一刻，我与你灵魂契合。

“开发环境使用es-module，直接跳过了webpack那繁琐的打包环节”，单这一点就使我醍醐灌顶。由于大家在开发环境使用的浏览器版本普遍较高，为了开发阶段的体验，完全可以省去一部分在生产环境才需要的降级工作，比如打包。这相当于告诉一个正在费力爬楼梯的人，不远处就有一部电梯！开发没必要与生产在构建策略上刻意保持一致，思路与格局就此打开。当然这也引起了不少人的顾虑与质疑，针对这个问题，你的作者是这样回复的:



我也认为问题不大，因为我与你的哥哥（Vuejs）是多年的好友，所以我会将这份信任复制到你身上，我相信你能平衡好这两者的一致性。

总之，我对你的第一印象非常好，甚至在脑海里已经计划好了使用你重构项目的RoadMap。

相处

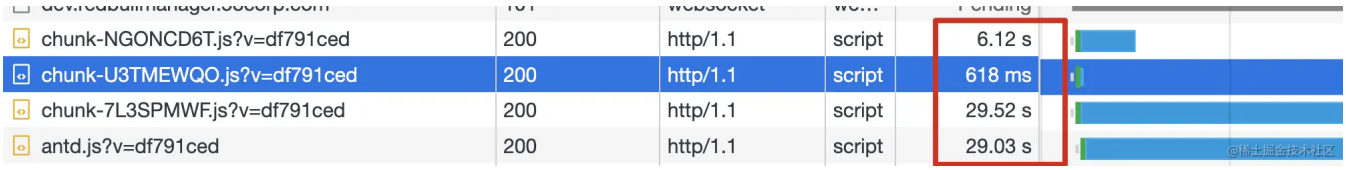
我的项目是一个使用react+ts+webpack维护的后台系统。框架层面使用vue与你应该是更搭的，毕竟你们是一家人。但你并没有拒绝我，你支持react，这体现了你的格局。

在重构过程中遇到的第一个问题，就是生态的不一致，这不是你的问题，但却是我必须考虑的适配成本。我们的项目重度依赖了webpack生态里一个负责条件编译的插件，用于在编译时区分新旧两套截然不同的权限版本。我在你的生态里也找到了对应的解决方案，但是，他们的条

件编译语法并不一致，所以，需要替换的代码块还是不少。这不是你的错，毕竟你文档的开篇就说了你与webpack大不相同，相当于两个世界。

后续遇到的问题，都是参考下文档就可以解决的磕磕碰碰。项目终于顺利跑起来了，而且启动项目的速度像你承诺的那般快，原本我还担心首次启动你会预构建npm依赖，花费的时间会比较久。我甚至已经想好了怎么安慰我自己，但你并没有给这个机会。我还知道，这是esbuild的功劳。我很激动，心里感叹，我果然没有看错你。但很快我发现了一件严重的事情。

项目虽然很快就启动了，但是当我在浏览器里访问页面地址时，长时间的一直处于请求“白板”状态。最后我发现从请求页面，到页面展现足足花了30s。。。这一度怀疑我的网络或者浏览器出故障了。



我开始去论坛上搜索，看有没有和我一样使用了vite但得到的是“负优化”（之前webpack的从启动项目到页面展示，一般在15s左右）的例子。我猜测大概率是我项目的问题，或者配置有不对的地方，毕竟你的npm下载量已是百万的量级，不至于和我玩低级的文字游戏。最后我从知乎上你的作者那里找到了答案：

尤雨溪

前端开发话题下的优秀答主

有些 webpack 项目切 vite 后觉得开发服务器页面首次打开慢。这种情况常见的原因是内部项目懒得做代码分割，所以一股脑整个项目几百上千个模块全都一起被加载了。解决方法就是至少在路由级别加上代码分割/懒加载，这样打开一个页面就只需要加载当前页面需要的模块。

发布于 2021-10-06 00:51

@稀土掘金技术社区

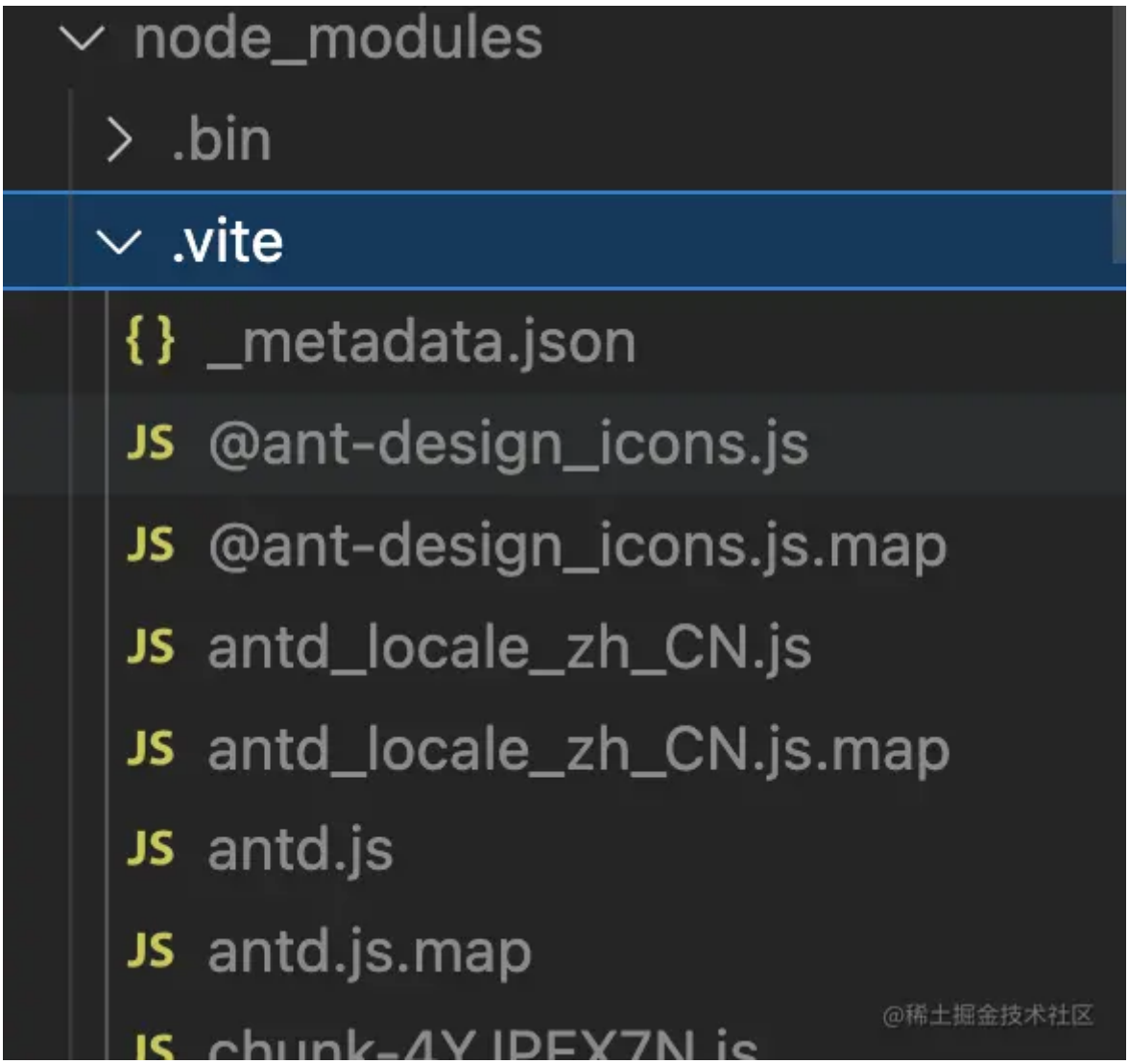
很不幸，我的项目就是你的作者口中的“懒得做代码分割的内部项目”。这让我有点羞愧，甚至觉得有些配不上你，但是我很快就安慰自己道：这项目也是接手的前任的烂摊子。自此让我明白了一件事：你并不是一个随意、大大咧咧的性格，想要发挥你的性能，需要顺着你的路子去走。无论如何，我找到了原因，而且相对也好改。用 `React.lazy(()=>import('./pages/xxx.tsx'))` 将路由懒加载一下即可。 与此同时，我回想起了你的特性之一“按需编译”。我是这样理解的，只有当前在浏览器真正需要访问的模块，你才会去编译转换。也就是说你并不是一个铺张浪费的人，而是喜欢节俭持家，量入为出。这点值得我的学习。

日子过的很快，我与你之间的新鲜感慢慢褪去。生活中难免有小插曲、小误会。比如我有一段时间，经常向周围的同事抱怨：你的HMR太难用了，动不动就刷新整个页面，这不是我理解的热更新，你只是鸡肋的帮我点了下浏览器的刷新按钮。我花费几分钟填写了一个长长的表单，验证某一项字段的功能是否正常。就因为不小心保存了一下代码，所有的页面状态，我辛苦填写的内容都被清空了。我开始对你产生抱怨，我心想以前使用webpack的时候，可没这么多小毛病。我后来偶然读到这篇文章才发现，我错怪你了。

> 即使没有写热更新代码也能够完成**React**热更新。 好啦！我们来看看插件更新规则。
>
> 这里是使用@vitejs/plugin-react更新规则,我们后面来说为什么是这样的。
>
> - 对于@vitejs/plugin-react实现的热更新来说,要想保存状态的更新(也就是说在执行保存操作之后,状态依旧存在,可以参照下面的效果)**只能是函数组件,并且hooks顺序不可改变,不可添加,不可删除**。以上操作都会导致热更新之后状态丢失。
> - 对于类组件来说,状态不可以保存,只能对类组件进行重新挂载。
> - 如果函数组件中包含了类组件,那么更新的基础点是重函数组件开始的,也就是说函数组件会重新调用并且获得新的**Virtual DOM**元素,然后向下比较在进行更新。
> - 如果类组件向上寻找,找不到函数组件,也就是说根组件就是类组件,那么就只能刷新网页了。
> - 综上所诉,更新的基准点是函数组件,且保存状态的更新要满足第一条的条件,类组件不可保存状态更新。
>
>

也就是说这不是你的错，这次误会是React官方的锅，是由于他们维护的react-refresh在处理函数组件、类组件时的差异导致的。大概意思是，支持保留状态的组件级热更新，对函数组件更加友好，类组件不支持。而我那个页面之前是用类组件实现的。从这次误会之后，我下定决心要将整个项目优化到至少符合你的期待的程度。我开始使用函数式组件重构旧页面，函数组件除了热更新支持力度高，还有一个好处，就是hook很容易从组件提取出去，成为可复用的自定义hook。我为此专门独立了一个npm包来维护这些公共hook。万万没想到，这件事引发了我对你最大的误会。

事情是这样的，我在hook库的开发环境使用rollup的watch模式来实时编译库文件，使用 `npm link package` 的方式，链接到业务项目里。link是成功的，我已经访问到了本地hook库的代码，但是当我修改了一些hook库的代码后，在你构建的项目里，始终看不到最新的效果。我开始了各种排查，检查是不是hook库的watch模式没生效，再重新link，还是不行。我开始四处寻找答案，终于在文档里，找到了相关的解释。导致这个问题的，有两个原因： 1.依赖预构建 2. 缓存。 为了兼容市面大量的非esm的包，和防止出现请求瀑布，这两件事决定了你必须要有预构建这一步，将源码里引用的npm包提前用esbuild打包好，并将他们缓存在 `node_modules/.vite` 目录下。



这意味着调试一个使用npm link链接过来的npm包，按常规方式:

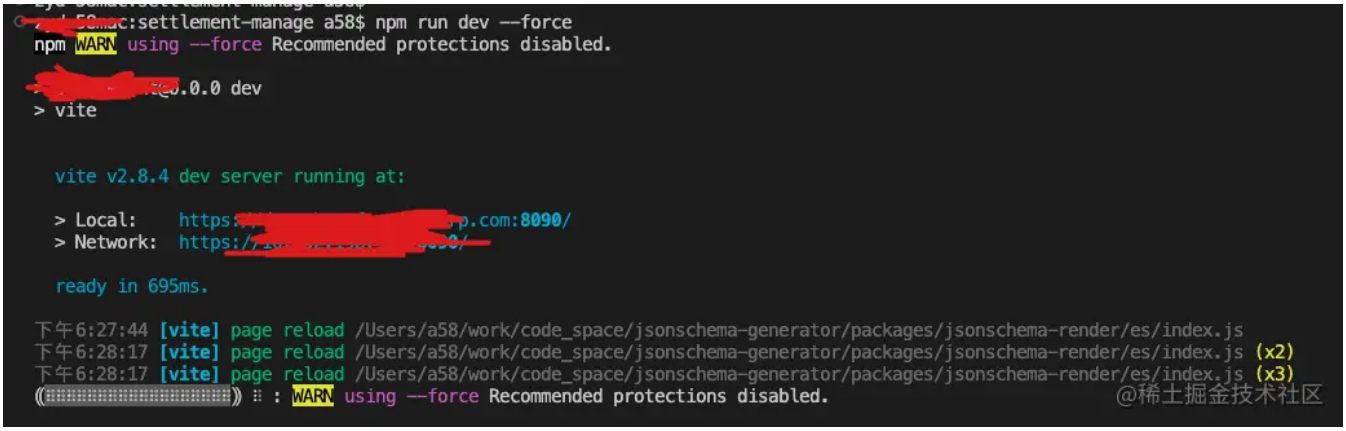
1. 更改链接库代码
2. 删除项目 `node_modules/.vite` 目录
3. npm run dev重启环境
4. 刷新浏览器

这意味着我即使想加一句console,也需要至少以上四个步骤。当然，我知道你有你的现实与苦衷，在文档里讲的很明白。

但确实影响到了我的包调试体验。我建议你可以出一个单独的库调试模式。在开启此模式后，在刷新页面时，所有依赖禁用读缓存，而是实时构建。这应该要慢很多，但我想，总比现在这样强。

2022-12-14日更新:

`npm run dev` 加 `--force` 可以解决这个问题，详见这里。哎，又是一场误会。



还有一点，我是那天看一个以《跨端》为主题的分享突然想到的。你好像在跨端构建这样的事情上，并不擅长。你所有的绝活，无论是 `type="module"` 的原生加载方式，还是基于浏览器缓存所做的页面请求"优化"，都强依赖现代浏览器。比如taro跨小程序、uniapp原生渲染这种场景也许直接用 `esbuild` 会比较好。

所以我觉得，"下一代web构建工具"的slogan好像更适合你。

现在

今晚，讲了许多我与你的故事。从一开始对你的狂热支持，到现在渐渐回归理性。使用得当，你依然相比webpack在构建web页面上快很多，但你并不适合所有的项目。比如你非常适合一些不需要很高的兼容要求，但对维护性有较高要求的中后台项目，因为这类的项目往往生命周期比较长。但在浏览器之外，比如编译跨端应用等场景，还是替代不了webpack。前一阵子，你与 `turbopack` 产生了一些争执，你的作者第一时间出来维护你，我对此表示羡慕。而且指出你们不是一个层面的东西，甚至将来 `turbopack` 的整体表现超过 `esbuild` ，可以考虑将它内置进vite用来预构建依赖，也就是成为vite的一部分。言下之意，你在设计层面是要“高”一层的，至少在web构建这个领域是这样。但是我觉得，抛开设计上的差异，你们要解决的问题其实是共同的，都是为了提升构建速度。那也就意味着你与他终有顶峰相遇的那一天。待那时，我希望你比现在更加稳定，更加全面。