使用vite作为项目的打包工具

背景：在ES Module诞生之前，前端开发者通常是通过手写Js代码来实现模块的封装。但由于Js的特性，所有使用Var声明的变量都是被挂载在windows全局变量上的，这就意味着随着项目的体积变大，页面的功能更加丰富。这样很容易造成变量的覆盖（因为你可能会不小心在后续的Js代码中重新声明并给这个已经存在的全局变量赋值），导致整个页面的崩溃。由此，前端程序员们最初想到的解决办法是使用立即执行函数封装独立的作用域的方式来解决这个问题。尽管如此，面对着页面交互时，频繁获取页面dom元素对象，并使用js原生给我们提供的api操作并完成页面的重新绘制，让前端程序员们还是有些头疼的。因此产生了一些批量获取并快速操作这些dom元素的库（例如jquery等），其实这些库就是一个js文件，最初使用这些库的方式非常简单，你只需要将这个库使用script标签加载进来，就可以在后续的script标签中使用它。这便是js模块化最初的模型了。但是这样的方式存在着几个问题：

1. 模块之间的引用关系不明显，可能会造成变量的覆盖，导致程序崩溃。
2. 由于网络波动的原因，有时即使你将需要的库放在页面的最上面加载，也不一定能够保证这个库在浏览器执行后续脚本之前加载完成。如果加载不成功，也许整个页面就会加载失败。

如何解决上面的问题呢？前端程序员们考虑可以将这些不同的js文件捆绑到一个js文件中，只要控制好他们的顺序（因为Js是一个从上到下执行的单线程语言），就一定不会报错了。

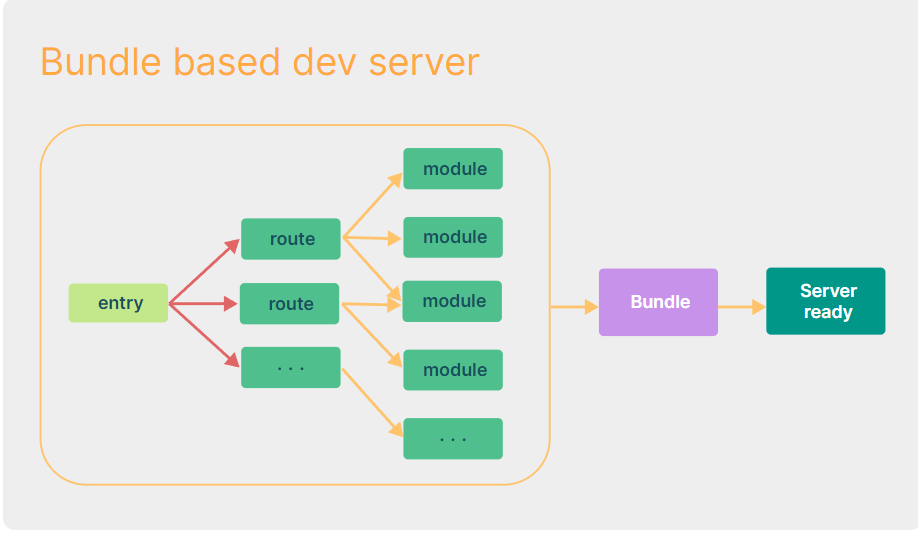
但是想象一下，如果一个页面有几十个或者上百个js文件，那么在以往的非单页面应用中由于每个页面完成的功能不一样，这样考虑到页面加载速度的原因，你也许需要封装出好多个bundle.js文件出来，每个页面对应一个bundle.js文件，这样繁琐的工作是让前端程序员们比较头疼的。所以在此背景下诞生了所谓的前端打包工具如：webpack、Rollup等，这些打包工具在帮助我们完成项目打包的同时，还帮助我们完成一些“页面加载速度优化“的工作。这里我们提到的vite就是这些工具的升级版，后续我会通过对比介绍的方式简单介绍为什么要使用vite这个项目打包工具。使用它的优势是什么。

为什么要使用vite?

在现代Js诞生之后（也就是es6之后），同时随着Js编译器的性能的提升(chrome的v8执行引擎的诞生)，Js已经可以构建大型的网页应用了，甚至已经走出了html页面的script标签，成为了一个前后端通吃的语言（nodeJs），甚至可以使用Js开发移动端和桌面端的应用，用Js开发的应用，所有平台通吃。真正实现了一套代码多个平台运行(如vscode,桌面端QQ等)！因此产生了许多大型的web项目，有时一个大型的web项目可能会有上千个modules。这时就出现了打包工具的性能瓶颈，严重影响了前端程序员的开发积极性。因为有时它们（webpack等）启动一个测试服务器并打开浏览器展示你所写的代码效果这个过程可能需要接近一分钟的时间！即使是你修改代码并保存查看效果这个过程也需要几秒钟的时间，这样的体验是不好的。使用Vite可以有效改善上述两种情况（1.项目测试启动时间过长2.在更新代码的时候热更新速度慢），接下来我们逐一论述vite在这两方面做了什么，对比传统的打包工具又有何优势：

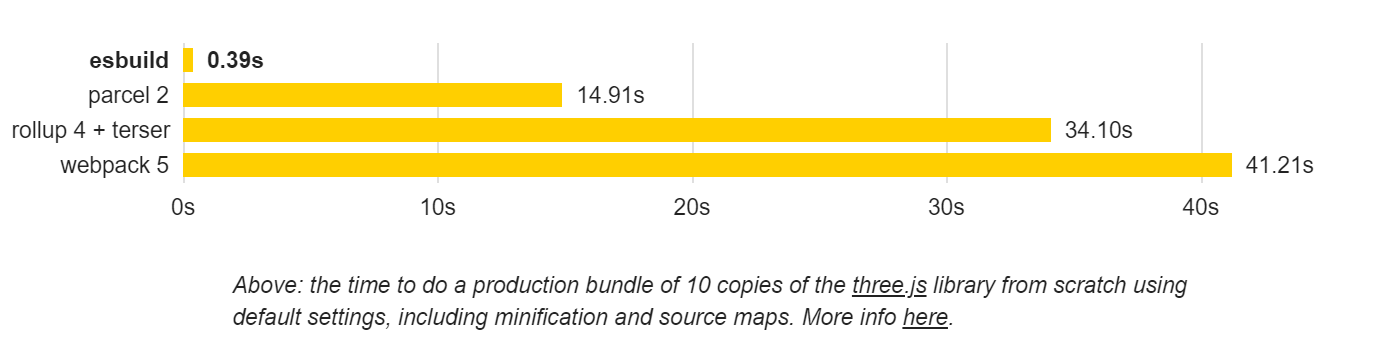
1. vite是如何缩短项目的启动时间的

在使用传统的打包工具时，启动这个项目一般是等待将所有的module打包到一个bundle.js文件完成后，dev server(测试服务器才正式准备好，才可以打开浏览器)

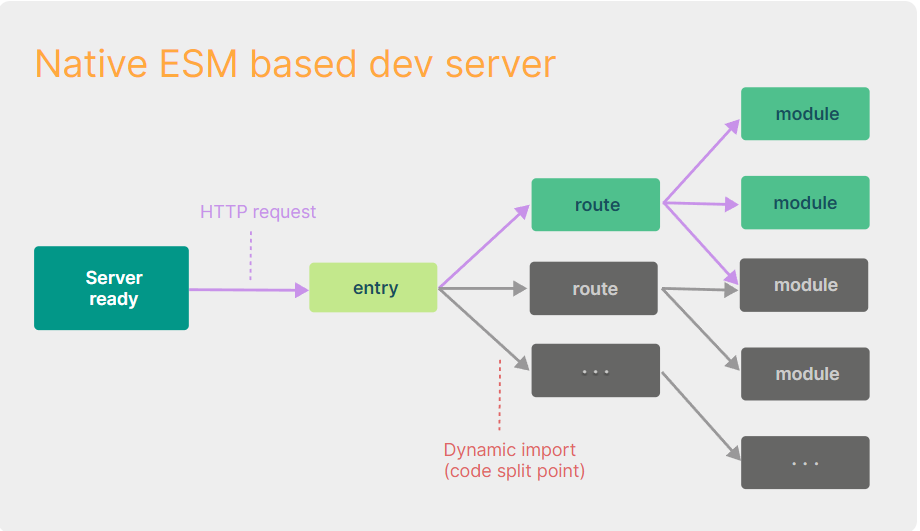


传统项目打包工具启动项目的流程

与传统打包工具相比较，vite启动项目的时间就快很多，这是因为vite将项目的代码分成两部分处理：依赖部分(dependencies)和核心部分(source code)。其中dependencies部分是指一些组件库，往往比较大，处理起来耗时比较长；source code则是指用户自己写的代码，这部分代码相对来说可能体积较小，但会经常修改。Vite将其分开处理，对于dependencies组件库的处理，vite采用全新的打包工具”esbuild”，这个打包工具的速度相当快，几乎是传统打包工具的10~100倍。



Esbuil打包three.js这个比较大型的组件库的耗时

同时为了提升项目启动的速度，vite不再将所有的module都加载（bundle）好后再启动测试服务器，而是直接启动服务器，当你在操作页面的过程中用到了那些modules的代码时，浏览器向vite启动的测试服务器请求，vite通过路由动态的向浏览器反馈。也就是说vite只需将你使用到的那部分modules经过处理后交给浏览器，而无需等待整个项目的所有modules都打包处理好后再启动测试服务器。

Vite启动测试服务器的流程图

1. vite是如何缩短项目的热更新时间的

由于传统的打包工具，是将所有的modules都bundle到一个js文件中的，所以随着项目的慢慢变大，引入的组件库慢慢变多，自己手写的代码也慢慢变多，加载速度肯定会慢慢变慢，虽然这些传统的打包工具在处理热更新的问题时，是通过部分修改内存中的bundle.js文件而不是再重新打包一份的方式来解决这个问题，但它仍需要重构整个bundle.js文件，这样相对来说是比较耗时的。相对来说Vite处理热更新的方式则简单快捷很多，Vite只将你更新过的那部分modules的HMR(Hot Module Replacement)boundary作废然后重新bundle即可。而且由于组件库的代码是不会变动的，vite在向浏览器发送了这部分代码后让浏览器对这部分代码进行强缓存，所以这部分代码不会占用测试路由的带宽。这样测试路由只需考虑如何将你用到的那部分代码反馈给浏览器并交由浏览器执行即可。这样其实是将部分性能开销转移给了浏览器，因为浏览器在执行你操作的那部分代码的时候需要通过路由向vite的测试服务器请求需要执行的代码。

资料来源：

https://vitejs.dev/guide/why.html

https://esbuild.github.io/