Vite编译原理

新型的前端构建工具——Vite。

构建工具是一个自动化将源代码生成可执行应用程序的程序。

Vite VS Webpack

Webpack

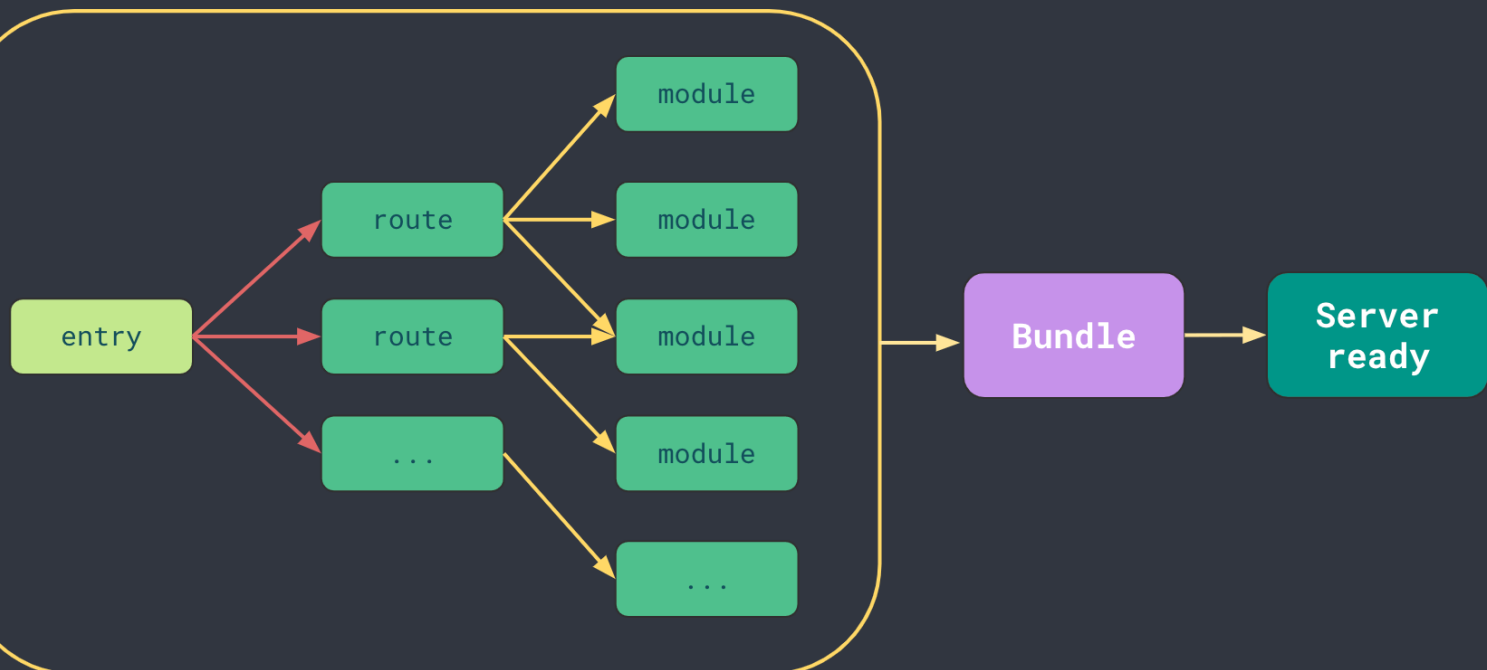
Webpack 是一个基于打包器的构建工具，同一个入口文件的代码会打包成一个 Bundle 文件。

Webpack**如何工作**

- 从一个入口文件开始，基于文件的import、export和require构建依赖树；
- 编译静态资源
- 经过各种优化打包生成 Bundle 文件 因为是整体编译打包，所以应用规模越大，启动和热更新代码越慢。这也是Webpack长期来的一个**痛点。**

> 当然，Webpack也有属于他的优势，可以关注一下Webpack5新加的模块联邦

Bundle based dev server



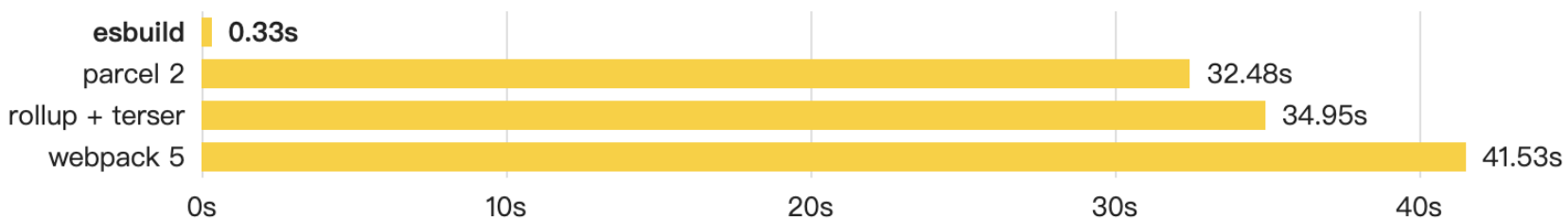
Vite

Vite 是旨在提升开发者体验的下一代 JavaScript 构建工具，核心借助了浏览器的原生 ES Modules 和使用了 esbuild 打包工具。

Vite如何工作

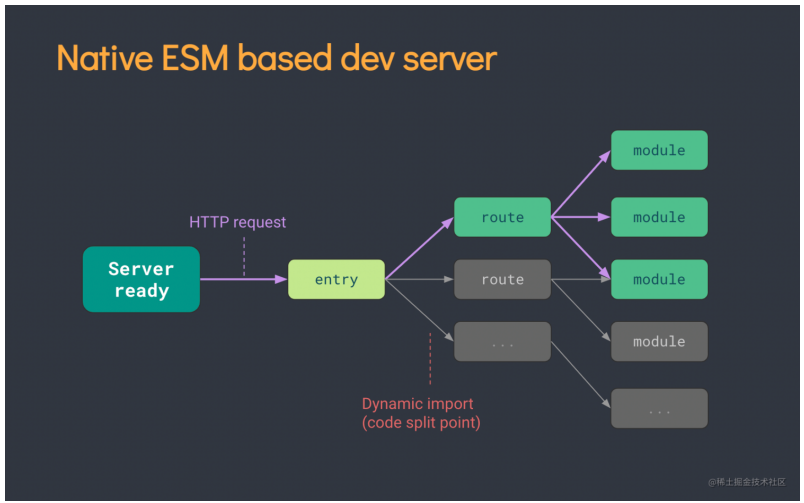
Vite将代码划分为依赖和源码，分别对其进行服务器启动时间的优化。

- 依赖模块：开发过程中基本不会变化，Vite 对依赖采用了 esbuild 预构建的方式，esbuild 使用 Go 编写，比以 JavaScript 编写的打包器预构建依赖快 10-100 倍；
- 源码模块，是用户自己开发的代码，会经常变动。Vite 在浏览器请求时按需转换并以原生 ESM 方式提供源码，让浏览器接管了打包程序的部分工作。



以上数据：分别是使用各工具的默认配置， 并从 0 开始构建 10 份 three.js 库的构建时间， 其中包括代码压缩以及 source map。 更多信息请查阅此处了解更多。

Vite 基于 ESM 按需提供源码文件，当一个文件被编辑后，Vite 只会重新编译并提供该文件。因此，无论项目规模多大，Vite 的热更新都可以保持快速更新。可以通过开发环境下，文件的hash值变化观察。



此外，Vite 合理利用浏览器缓存来加速页面加载，源码模块请求根据 304 Not Modified 进行协商缓存；依赖模块请求通过 Cache-Control: max-age=31536000,immutable 进行强缓存，因此一旦缓存，不会再次请求。

依赖预构建

Vite 在首次启动时，会进行依赖预构建。依赖预构建有两个目的：

1. **CommonJS 和 UMD 兼容性**: 开发阶段中，Vite 的开发服务器将所有代码视为原生 ES 模块。因此，Vite 必须先将作为 CommonJS 或 UMD 发布的依赖项转换为 ESM。
2. **性能：** Vite 将有许多内部模块的 ESM 依赖关系转换为单个模块，以提高后续页面加载性能。比如，lodash-es 拥有超过 600 个内部模块，当 `` `import { debounce } from 'lodash-es'; ` `` 时，浏览器会同时发起超过 600 个请求，并行请求过多将会显著影响页面加载性能。因此预构建将 lodash-es 视为一个模块，浏览器只需要发起一个请求。

```
export {a} from './a'
export {b} from './b'

export default 0
```

文件系统缓存

Vite 会将预构建的依赖缓存到 node_modules/.vite。它根据几个源来决定是否需要重新运行预构建步骤:

- 包管理器的 lockfile 内容，例如 package-lock.json，yarn.lock，pnpm-lock.yaml，或者 bun.lockb
- 补丁文件夹的修改时间
- 可能在 vite.config.js 相关字段中配置过的
- NODE_ENV 中的值

只有在上述其中一项发生更改时，才需要重新运行预构建。
如果出于某些原因，你想要强制 Vite 重新构建依赖，你可以用 --force 命令行选项启动开发服务器，或者手动删除 node_modules/.vite 目录。

实践

顺带介绍几个 Babel工具

- @babel/parser : 将 js 代码 ------->>> AST 抽象语法树;
- @babel/traverse 对 AST 节点进行递归遍历;
- @babel/types 对具体的 AST 节点进行进行修改;
- @babel/generator : AST 抽象语法树 ------->>> 新的 js 代码;