vue cli 从入门到精通

cli 相关概念及知识

cli 是一种通过命令行来交互的工具应用，全称时 Command Line Interface。比较常见的就是 create-react-app，vue-cli 等，他们都能够将一段 js 脚本，通过封装为可执行代码的形式，进行一些操作。

使用 cli 之后呢，能快速的创建一些我们业务中的样板文件，比如快速创建一个项目内容，配置公共的 eslint、webpack 等等配置工具。

在封装这些内容之前，我们需要使用如下的几个库：

- commander：命令行中的参数获取
- inquirer：命令行的表单
- chalk：命令行中的可变颜色效果
- clui：命令行中的 loading 效果
- child_process：node 原生模块，提供一些方法让我们能够执行新的命令

child_process 中有一些方法，比如 exec 等，`exec`方法用于新建一个子进程，然后缓存它的运行结果，运行结束后调用回调函数。

我们这里可以使用 execSync，它能够执行一些我们 linux 中的命令。

commander 对命令行进行了解析，可以让我们比较方便的进行命令行参数的获取，读取和解析

chalk 对应的是命令行文字颜色的更改

clui 是一个命令行中展示 loading 效果的库

@vue/cli 相关使用

Vue CLI 是一个基于 Vue.js 进行快速开发的完整系统，提供:

- 通过 @vue/cli 搭建交互式的项目脚手架。
- 通过 @vue/cli + @vue/cli-service-global 快速开始零配置原型开发。
- 一个运行时依赖 (@vue/cli-service) 一个丰富的官方插件集合，集成了前端生态中最好的工具。
- 一套完全图形化的创建和管理 Vue.js 项目的用户界面。

我们能够通过 `npm install -g @vue/cli` 来进行安装，安装成功之后，我们就能使用 vue 这个命令。

```
 npm install -g @vue/cli-service-global

# 启动一个服务器
vue serve xxx.vue
# 将目标文件构建成一个生产环境的包
vue build xxx.vue


# 创建一个由 vue-cli-service 提供支持的新项目
vue create mytest
```

你会被提示选取一个 preset。你可以选默认的包含了基本的 Babel + ESLint 设置的 preset，也可以选“手动选择特性”来选取需要的特性。这个默认的设置非常适合快速创建一个新项目的原型，而手动设置则提供了更多的选项，它们是面向生产的项目更加需要的。

```
# 创建一个由 vue-cli-service 提供支持的新项目
vue create mytest

# 使用 UI 界面
vue ui
```

插件和预设配置插件

Vue CLI 使用了一套基于插件的架构，package.json中依赖都是以 @vue/cli-plugin- 开头的。插件可以修改内部的 webpack 配置，也可以向 vue-cli-service 注入命令。在项目创建的过程中列出的特性，绝大部分都是通过插件来实现的。

预设配置

Vue CLI 预设配置是一个包含创建新项目所需的预定义选项和插件的 JSON 对象，让用户无需在命令 提示中选择它们。在 vue create 过程中保存的预设配置会被放在你的 home 目录下的一个配置文件中 (~/.vuerc)。你可以通过直接编辑这个文件来调整、添加、删除保存好的配置。

这里有一个预设配置的示例:

```
# 在现有的项目中安装插件
vue add eslint
```

```
{
 "useConfigFiles": true,
  "router": true,
 "vuex": true,
 "cssPreprocessor": "sass",
 "plugins": {
    "@vue/cli-plugin-babel": {},
    "@vue/cli-plugin-eslint": {
      "config": "airbnb",
      "lintOn": ["save", "commit"]
    }
  }
}
```

CLI 服务

在一个 Vue CLI 项目中，@vue/cli-service 安装了一个名为 `vue-cli-service` 的命令。你可以在 npm

scripts 中以 `vue-cli-service`、或者从终端中以 `./node_modules/.bin/vue-cli-service` 访问这个命令。

```
{
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build"
   }
}

npm run serve
```

开发浏览器兼容性

一个默认的 Vue CLI 项目会使用 @vue/babel-preset-app，它通过 @babel/preset-env 和 browserslist 配置来决定项目需要的 polyfill。

默认情况下，它会把 `useBuiltIns: 'usage'` 传递给 `@babel/preset-env`，这样它会根据源代码中出现的 语言特性自动检测需要的 polyfill。这确保了最终包里 polyfill 数量的最小化。然而，这也意味着如果其 中一个依赖需要特殊的 polyfill，默认情况下 Babel 无法将其检测出来。

如果有依赖需要 polyfill，你有几种选择: 如果该依赖基于一个目标环境不支持的 ES 版本撰写: 将其添加到 vue.config.js 中的

transpileDependencies 选项。这会为该依赖同时开启语法语法转换和根据使用情况检测 polyfill。

如果该依赖交付了 ES5 代码并显式地列出了需要的 polyfill: 你可以使用 @vue/babel-preset-app 的 polyfills 选项预包含所需要的 polyfill。注意 es6.promise 将被默认包含，因为现在的库依赖 Promise 是非常普遍的。

```
// babel.config.js

module.exports = {
  presets: [
    [
      '@vue/app',
      {
        polyfills: [
          'es6.promise',
          'es6.symbol'
        ]
      }
    ]
  ]
}
```

如果该依赖交付 ES5 代码，但使用了 ES6+ 特性且没有显式地列出需要的 polyfill (例如 Vuetify): 请使用 useBuiltIns: 'entry' 然后在入口文件添加 import '@babel/polyfill'。这会根据 browserslist 目 标导入所有 polyfill，这样你就不用再担心依赖的 polyfill 问题了，但是因为包含了一些没有用到的 polyfill 所以最终的包大小可能会增加。

现代模式

有了 Babel 我们可以兼顾所有最新的 ES2015+ 语言特性，但也意味着我们需要交付转译和 polyfill 后的包以支持旧浏览器。这些转译后的包通常都比原生的 ES2015+ 代码会更冗长，运行更慢。现如今绝大多数现代浏览器都已经支持了原生的 ES2015，所以因为要支持更老的浏览器而为它们交付笨重的 代码是一种浪费。

Vue CLI 提供了一个“现代模式”帮你解决这个问题。以如下命令为生产环境构建:

```
vue-cli-service build --modern
```

Vue CLI 会产生两个应用的版本：

- 一个现代版的包，面向支持 ES modules 的现代浏览器
- 另一个旧版的包，面向不支持的旧浏览器

HTML Preload

preload 是一个 HTML5 的新特性，是一个新的标签，对于浏览器加载来说，对于主资源HTML和CSS的优先级最高，其他资源优先级不统一。我们使用 preload 属性，可以让支持的浏览器提前加载资源，但加载时并不执行，等待需要时才进行执行。

这样做的好处就是我们可以将加载和执行分离开，同时也可以控制提前加载一些大型文件，防止使用时获取的页面闪烁。

我们就可以用来指定页面加载后很快会被用到的资源，所以在页面加载的过程中，我们希望在浏览器开始主体渲染之前尽早 preload。

默认情况下，一个 Vue CLI 应用会为所有初始化渲染需要的文件自动生成 preload 提示。 这些提示会被 **@vue/preload-webpack-plugin** 注入，并且可以通过 **chainWebpack** 的

`config.plugin('preload')` 进行修改和删除。

Prefetch

是一种 resource hint，用来告诉浏览器在页面加载完成后，利用空闲时间提前获取用户未来可能会访问的内容。

默认情况下，一个 Vue CLI 应用会为所有作为 **async chunk** 生成的 JavaScript 文件 (通过动态 `import()` 按需 code splitting 的产物) 自动生成 prefetch 提示。

这些提示会被 **@vue/preload-webpack-plugin** 注入，并且可以通过 **chainWebpack** 的 `config.plugin('prefetch')` 进行修改和删除。

举个例子：

```
// vue.config.js

module.exports = {
  chainWebpack: config => {
   // 移除 prefetch 插件 config.plugins.delete('prefetch')

   // 或者
   // 修改它的选项：
    config.plugin('prefetch').tap(options => {
      options[0].fileBlacklist = options[0].fileBlacklist || []
      options[0].fileBlacklist.push(/myasyncRoute(.)+?.js$/)
      return options
    })
  }
}
```

当 prefetch 插件被禁用时，你可以通过 webpack 的内联注释手动选定要提前获取的代码区块:

webpack 的运行时会在父级区块被加载之后注入 prefetch 链接。

`import(/* webpackPrefetch: true */ './someAsyncComponent.vue')`

处理静态资源

静态资源可以通过两种方式进行处理:

- 在 JavaScript 被导入或在 template/CSS 中通过相对路径被引用。这类引用会被 webpack 处理。

当你在 JavaScript、CSS 或 **.vue** 文件中使用相对路径 (必须以 . 开头) 引用一个静态资源时，该资源将会被包含进入 webpack 的依赖图中。在其内部，我们通过 **file-loader** 用版本哈希值和正确 的公共基础路径来决定最终的文件路径，再用 url-loader 将小于 4kb 的资源内联，以减少 HTTP 请求的数量。

- 放置在 public 目录下或通过绝对路径被引用。这类资源将会直接被拷贝，而不会经过 webpack 的处理。

CSS 相关

Vue CLI 项目天生支持 PostCSS、CSS Modules 和包含 Sass、Less、Stylus 在内的预处理器。

所有编译后的 CSS 都会通过 css-loader 来解析其中的 `url()` 引用，并将这些引用作为模块请求来 处理。这意味着你可以根据本地的文件结构用相对路径来引用静态资源。

你可以在创建项目的时候选择预处理器 (Sass/Less/Stylus)。

如果当时没有选好，内置的 webpack 仍然会被预配置为可以完成所有的处理。

webpack 相关

调整 webpack 配置最简单的方式就是在 vue.config.js 中的 configureWebpack 选项提供一个对象，该

对象将会被 webpack-merge 合并入最终的 webpack 配置。

```
// vue.config.js

module.exports = {
      configureWebpack: {
        plugins: [
          new MyAwesomeWebpackPlugin()
        ]
} }
```

构建目标

当你运行 `vue-cli-service build` 时，你可以通过 `--target` 选项指定不同的构建目标。应用模式是默认的 模式。在这个模式中:

index.html 会带有注入的资源和 resource hint 第三方库会被分到一个独立包以便更好的缓存 小于 4kb 的静态资源会被内联在 JavaScript 中 public 中的静态资源会被复制到输出目录中

部署

如果你独立于后端部署前端应用——也就是说后端暴露一个前端可访问的 API，然后前端实际上是纯 静态应用。那么你可以将 dist 目录里构建的内容部署到任何静态文件服务器中，但要确保正确的 publicPath。

vue cli 源码解析

1. @vue/cli 生成的 vue 命令
2. @vue/cli vue 命令下的 create 方法
3. @vue/cli-service 的 serve 命令
4. @vue/cli-service 的 build 命令
5. @vue/cli-plugin-eslint 的 eslint 插件
6. @vue/cli add 命令增加一个插件