

首屏加载优化

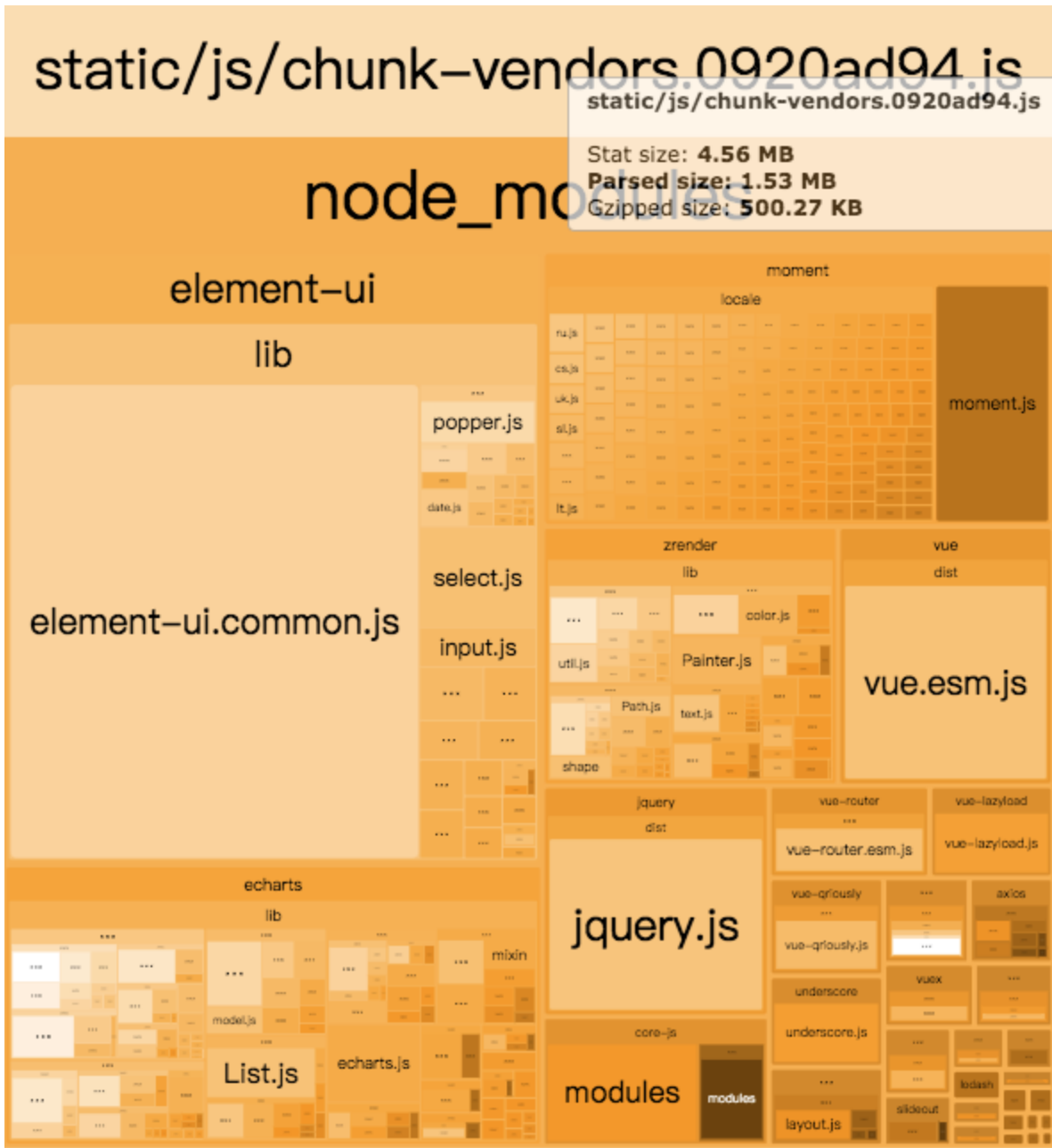
问题

单页面应用的一个问题就是首页加载东西过多，加载时间过长。特别在移动端，单页面应用的首屏加载优化更是绕不开的话题。下面我会写出我在项目中做的一些优化，希望大家能够相互讨论，共同进步。

我的项目vue-cli3构建的，vue+vue-router+vuex，UI框架选用 element-ui，ajax方案选用 axios，服务器使用Nginx。用到的这些技术都是现在用的比较广泛的，看到这篇文章，我估计你和我用的技术应该差不多。

第一步：webpack-bundle-analyzer 分析

首页我们来看看没有经过任何优化的打包分析，vue-cli3的项目直接vue-cli-service build --report就会生成一个report.html，打开这个html就能看到，不是vue-cli3的项目需要自行安装这个插件，参考链接，[点击](#)。



如上图所示在vendor比较大的文件有element，moment，echart，还有jquery，然后还有一些没见过的vue-qriousy这些组件，接下来我们来一步一步让vendor变小

第二步：初步优化

1. 仔细考虑组件是否需要全局引入

在我们的main.js，我发现有很多组件被全局引入，其中有些组件只有1，2个页面用到，这些组件不需要全部引入

```
import ImageComponent from 'COMMON/imageComponent'
```

```
import InfiniteLoading from 'COMMON/infiniteLoading'
```

```
import SearchDialog from 'COMMON/SearchDialog'
```

```
import BasicTable from 'COMMON/BasicTable'
```

```
import VueQriously from 'vue-qriously'
```

```
Vue.use(ImageComponent)
```

```
Vue.use(InfiniteLoading) // 可以去除
```

```
Vue.use(SearchDialog) // 可以去除
```

```
Vue.use(BasicTable) // 可以去除
```

```
Vue.use(VueQriously) // 可以去除
```

上面一段代码是我们main.js中的代码，其中ImageComponent是用来处理图片的，用到的页面很多，其他的组件都只要较少的页面用到，我们在main.js中删除，移到具体的页面中去。

2. 手动引入 ECharts 各模块

默认引入 ECharts 是引入全部的``import * as ECharts from 'echarts'``我们只需要部分组件，只需引入自己需要的部分。参考地址，[点击](#)

```
import VueECharts from 'vue-echarts/components/ECharts.vue'
```

```
import 'echarts/lib/chart/line'
```

```
import 'echarts/lib/chart/bar'
```

```
import 'echarts/lib/chart/pie'
```

```
import 'echarts/lib/component/title'
```

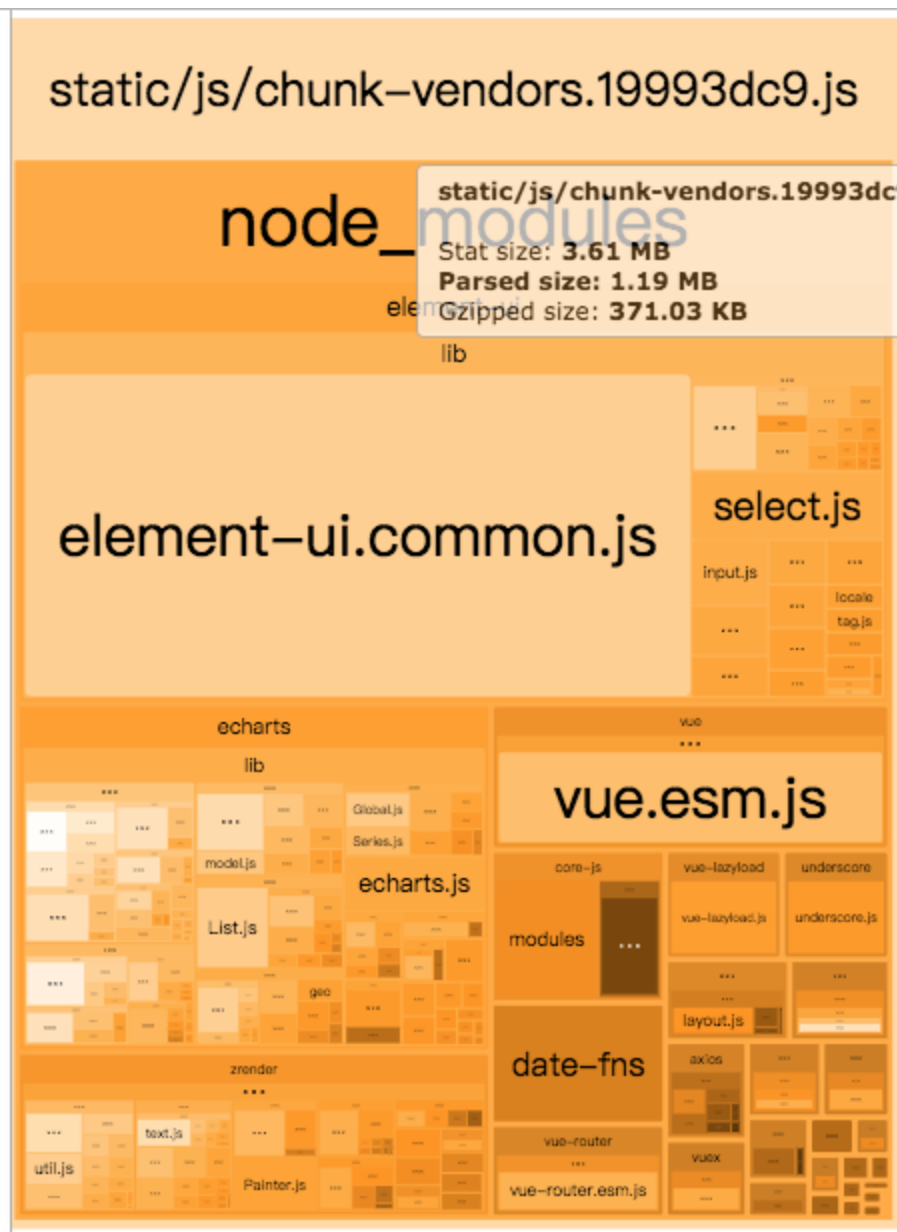
```
import 'echarts/lib/component/tooltip'
```

```
import 'echarts/lib/component/legend'
```

```
import 'echarts/lib/component/markPoint'
```

3.使用更轻量级的工具库

moment是处理时间的标杆，但是它过于庞大且默认不支持tree-shaking，而且我们的项目中只用到了moment(), format(), add(), subtract()等几个非常简单的方法，有点大材小用，所以我们用 [date-fns](#) 来替换它，需要什么方法直接引入就行。



经过上面的三步初步优化，我们可以看到vendor.js变小了很多，去除了moment，我们项目之前echart就是按需加载的。

第三步：CDN优化

进过上面的优化，发现 Vue 全家桶以及 ElementUI 仍然占了很大一部分 vendors 体积，这部分代码是不变的，但会随着每次 vendors 打包改变 hash 重新加载。我们可以使用 CDN 剔除这部分不经常变化的公共库。我们将vue, vue-router, vuex, axios, jquery, underscore, 使用CDN资源引入。国内的CDN服务推荐使用 [BootCDN](#)

1.首先我们在index.html中，添加CDN代码

```
...  
<link href="https://cdn.bootcss.com/element-ui/2.7.2/theme-chalk/index.css"  
rel="stylesheet">  
</head>
```

```

<body>
  <div id="app"></div>
  <script src="https://cdn.bootcss.com/vue/2.6.10/vue.min.js"></script>
  <script src="https://cdn.bootcss.com/vuex/3.1.0/vuex.min.js"></script>
  <script src="https://cdn.bootcss.com/vue-router/3.0.4/vue-router.min.js"></script>
  <script src="https://cdn.bootcss.com/axios/0.18.0/axios.min.js"></script>
  <script src="https://cdn.bootcss.com/element-ui/2.7.2/index.js"></script>
  <script src="https://cdn.bootcss.com/jquery/3.4.0/jquery.min.js"></script>
  <script src="https://cdn.bootcss.com/underscore.js/1.9.1/underscore-min.js"></script>
</body>
</html>

```

2.在vue.config.js中加入webpack配置代码，关于webpack配置中的externals，请参考[地址](#)

```

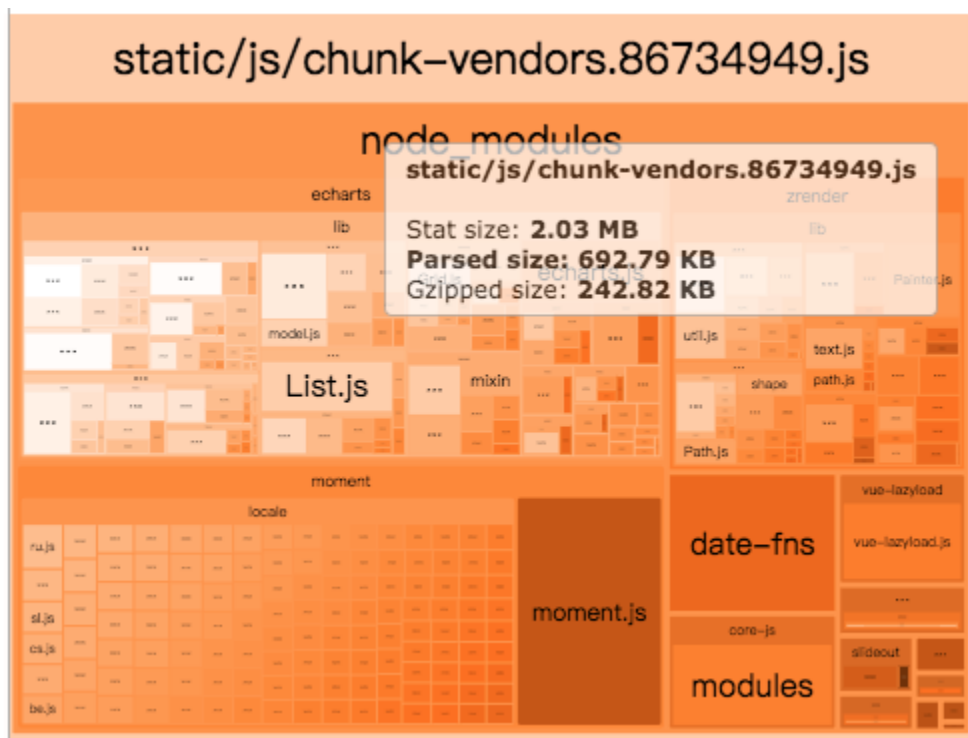
configureWebpack: {
  externals: {
    'vue': 'Vue',
    'vue-router': 'VueRouter',
    'vuex': 'Vuex',
    'element-ui': 'ELEMENT',
    'axios': 'axios',
    'underscore' : {
      commonjs: 'underscore',
      amd: 'underscore',
      root: '_'
    },
    'jquery': {
      commonjs: 'jQuery',
      amd: 'jQuery',
      root: '$'
    }
  },
}

```

3. 去除vue.use相关代码

需要注意的是，通过 CDN 引入，在使用 VueRouter Vuex ElementUI 的时候要改下写法。CDN会把它们挂载到window上，因此不再使用Vue.use(xxx)

也不在需import Vue from 'vue', import VueRouter from 'vue-router' 等。



剔除全家桶和Element-ui等只有，剩下的需要首次加载 vendors 就很小了。





使用 CDN 的好处有以下几个方面

- (1) 加快打包速度。分离公共库以后，每次重新打包就不会再把这些打包进 vendors 文件中。
- (2) CDN减轻自己服务器的访问压力，并且能实现资源的并行下载。浏览器对 src 资源的加载是并行的(执行是按照顺序的)。

第四步：检查Nginx 是否开启 gzip

如下图所示，开启了gzip后js的大小比未开启gzip的js小2/3左右，所以如果没开启gzip，感觉我们做的再多意义也不大，如何看自己的项目有没有开启gzip，如下图所示，开启了gzip，在浏览器的控制台Content-Encoding一栏会显示gzip，否则没有。Nginx如果开启gzip，请自行搜索，或者叫服务端来开启。

File	Size	Gzipped
dist/static/js/chunk-vendors.0920ad94.js	1570.15 kb	501.47 kb
dist/static/js/chunk-c0a63ee6.5c81b5ea.js	492.71 kb	143.17 kb
dist/static/js/chunk-f1acea9c.046a852d.js	411.99 kb	139.24 kb
dist/static/js/chunk-31ff4981.76e6a10e.js	242.15 kb	23.52 kb
dist/static/js/chunk-cde62ba0.352db918.js	204.58 kb	19.55 kb

Name	Status	Type	Initiator	Size	Time	Content-Encoding
 app.6e8fd98b.js	200	sc...	(index)	101 KB	765 ms	
 chunk-vendors.378c...	200	sc...	(index)	719 KB	2.36 s	
 vue.min.js	200	sc...	(index)	33.7 KB	255 ms	gzip
 vuex.min.js	200	sc...	(index)	3.3 KB	101 ms	gzip

第五步：检查路由懒加载

路由组件如果不按需加载的话，就会把所有的组件一次性打包到app.js中，导致首次加载内容过多，vue官方文档中也有提到，[地址](#)。

```
{
  name: 'vipBoxActivity',
  path: 'vipBoxActivity',
  component: resolve() {
    require(['COMPONENTS/vipBox/vipBoxActivity/main.vue'], resolve)
  }
},
{
  path: 'buyerSummary',
  name: 'buyerSummary',
  component: () => import('VIEWS/buyer/buyerSummary/index'),
},
```

上面的两种引入组件的方法都是正确的，都能实现路由的懒加载。

最后

最后我们可以发现vendor.js的大小减少了很多。其中第一步到第三步我们项目中都没做，第四步和第五步我们做了。如果读者你没做，一定要注意了。最后希望这篇文章能够对大家有一点点帮组

图片优化：

// 小于8K的图片将直接以base64的形式内联在代码中，可以减少一次http请求。

// 大于8k的呢?则直接file-loader打包，这里并没有写明file-loader.但是确实是需要安装,否则会有问题.而name也是file-loader的属性. 重复一次 必须安装file-loader

```
{
  test: /\. (woff|woff2|eot|ttf|svg|jpg|png|gif)\??.*$/,
  loader: 'url-loader',
  query: {
    // 图片大小限制 单位b
    limit: 8192,
    // 生成的文件的存放目录
    name: 'resource/[name].[ext]'
  }
},
```