代码规范

命名规范

变量：CamelCase

常量：使用大写字母和下划线来组合命名，下划线用以分割单词。

组件：声明（PascalCase）、使用（KebabCase）

函数：CamelCase，统一使用动词或者动词 + 名词形式

Prop：声明（CamelCase）、使用（KebabCase）

目录结构

```
src                                 源码目录
|-- api                             模块接口
|-- assets                          静态资源
|-- components                      全局组件
|-- icons                           图标资源
|-- libs                            外部引用的插件
|-- pages                           多页面目录，构建MPA
|-- router                          路由
|-- store                           vuex
|-- utils                           全局工具
|-- vant                            UI组件
|-- views                           视图目录
|   |-- article                     视图模块名
|   |-- |-- components             模块级组件
|   |-- |-- articleItem.vue        模块页面
|   |-- |-- articleList.vue        模块页面
```

组件文件

- 多个 attribute 的元素应该分多行撰写，每个 attribute 一行。
- 组件选项的顺序

```
- components
- filters
- props
- data
- computed
- watch
- created
- mounted
- methods
```

- prop 的定义应该尽量详细，至少需要指定其类型。
- 在组件上总是必须用 key 配合 v-for，以便维护内部组件及其子树的状态。
- 避免 v-if 和 v-for 用在一起。
- 为组件样式设置作用域 scoped，或遵循 BEM 的策略。

注释规范

- 代码注释量的占比至少要达到**30%**以上。
- VS Code 插件(koroFileHeader)在 settings.json 中添加注释模版

```
"fileheader.customMade": {
  "Descripttion":"",
  "Author":"wangmiao@mochasoft.com.cn",
  "Date":"Do not edit",
  "LastEditors":"wangmiao@mochasoft.com.cn",
  "LastEditTime":"Do not Edit"
},
"fileheader.cursorMode": {
  "description":"",
  "param":"",
  "return":""
},
"fileheader.configObj": {
  "autoAdd": false
}
```

其他规范

1. 关于单引号和双引号的使用问题：
    - 在 HTML 中统一使用双引号；
    - 在 JSON 中统一使用双引号；
    - 在 JavaScript 中统一使用单引号，拼接字符串请使用模板字符串(ES6)；
2. 不要自己擅自造轮子！常用 JavaScript 轮子首选：Moment.js(Day.js)、Lodash。
3. 通过 babel-plugin-transform-remove-console 插件，在构建生产环境时，移除 console。
4. 严格管控 dependencies 与 devDependencies 中依赖包的安装。
5. CSS 要求使用 Less 或者 Sass。

用户体验

路由过渡效果

基于当前路由与目标路由的变化关系，通过 transition 组件动态设置过渡效果，也可以使用 Vant 内置的动画。

请求加载动画

- 通过 Vant 组件 Toast 实现，在 Axios 实例对象的请求/响应拦截器中全局控制。
- 通过 Vant 组件 Loading 实现，在模块组件的生命周期中自行控制。

骨架屏

通过 Vant 组件 Skeleton 实现，可配合 请求加载动画 一并使用。

移动端适配

- postcss-pxtorem：把 px 单位换算成 rem 单位
- amfe-flexible：移动适配方案，自动计算 Html 的 font-size 字体的大小

构建MPA

MPA：多页面应用，也可以理解为多入口应用。

通常我们用 Vue 开发的项目基本上都是 SPA，但是在某些业务场景下 SPA 并不适用，比如移动办公的业务模块与下载页面，如果打包成 SPA 的话，用户在第一次访问下载页面的时候，势必会加载 业务模块的 js 文件，导致白屏时间过长，这显然是不合理的。所以 我们需要将 业务模块 与 非业务模块 拆分成两个独立的入口，各自拥有独立的 router、store 等。

性能优化

打包分析

通过插件(webpack-bundle-analyzer)分析打包文件。

路由懒加载

结合 Vue 的异步组件和 Webpack 的代码分割功能，轻松实现路由组件的懒加载，如果项目使用的是 Babel，则需要添加 syntax-dynamic-import 插件，才能使 Babel 可以正确地解析语法。

资源预读取(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。

为了提升首屏的渲染速度，可以移除 prefetch 插件。

Vant 组件按需引入

依赖 babel-plugin-import 插件，它会在编译过程中将 import 的写法自动转换为按需引入的方式。

Lodash 按需引入与代码压缩

按需引入

依赖 babel-plugin-lodash 插件，对引用的 lodash 进行类似 tree shaking 的操作，这样就可以去除未使用的 lodash 代码片段。

代码压缩

依赖 lodash-webpack-plugin 插件，可以进一步压缩 lodash。

Gzip 压缩

通过 compression-webpack-plugin 插件可以将大文件压缩成 gzip 的格式，服务器端（nginx）也需要开启 gzip 压缩的功能。

图片压缩

通过 image-webpack-loader 压缩图片资源。

CDN 资源

我们作下数据分析，引用的 CDN 资源大小如下：

1. vue.min.js 34.9k
2. vuex.min.js 4.6k
3. vue-router.min.js 10.5k
4. axios.min.js 5.6k
5. dayjs.min.js 3.5k
6. js.cookie.min.js 1.6k

以项目代码为例，打包后 chunk-vendors 文件大小的数据对比：

File	Size	Gzipped
chunk-vendors.0d8eedf9.js（本地集成打包）	198k	68k
chunk-vendors.825fb3ea.js（引入 cdn 资源：1～4）√	46k	16k
chunk-vendors.7102db14.js（引入 cdn 资源：1～6）	44k	16k

我们可以看到 gzip 压缩后，浏览器加载的资源总量并没有太大变化，**所以要根据实际情况引用 cdn 资源。**

> 优点

- 可以降低 chunk-vendors.js 文件的大小。
- 浏览器可以多线程异步加载 chunk-vendors.js 与 CDN 资源，加快首屏渲染的速度。

> 缺点

- 需要维护 CDN 资源的版本。
- 需要考虑 CDN 网站的稳定性。

代码分割

webpack4.x 中，依赖 SplitChunksPlugin 将共享代码块单独打包，通过代码分离的方式，可以减少首屏的代码体积，从而提高首屏的加载速度。

另外，在我们的项目中，可能会用到很多的第三方库（比如 lodash 、dayjs 等），而往往这些第三方依赖库的代码一般很少变化，因此，很适合把第三方依赖库单独分离成一个包，并且包名包含 hash，这样的好处在于，可以配合浏览器 http 的缓存机制，实现对相关资源包的长缓存，从而优化性能。

一般需要代码分割的场景有：

- 分离业务代码和第三方依赖。
- 分离首次加载和异步加载的代码。
- 分离业务代码和业务的公用代码。