本系统采用React + React Router + Redux等技术构建，主体工程结构遵循通用的规范

采用react-app-rewired 覆盖create-react-app脚手架的配置，同时引入antd、less等、通过babel-import-plugin实现按需加载

工程结构如下：

```

Brain

├── doc --- 系统说明文档

├── public --- 页面入口目录，公共静态资源，如图片、js库等

│ └── images --- 图片

│ └── lib --- js库

│ └── mock --- 模拟数据

│ └── index.html --- 系统唯一的页面入口

└── src --- 代码入口（类似dva目录结构，公认的结构规范）

└── common --- 全局通用的逻辑代码、资源文件等

└── components --- 与业务无关的可反复使用的组件

└── redux --- redux状态管理容器相关内容

└── routes --- React Router路由组件，也是系统的模块，与业务强关联的组件也应存放于此

└── Blank --- 空白页面，可以看作是一个页面模板

└── config.json --- 业务模块配置文件

└── common --- routes下通用的资源文件、业务相关的组件、逻辑代码等

└── Home --- 系统首页

└── Traffic --- 智慧交通页面

└── index.js --- 路由入口

└── services --- 业务逻辑

└── utils --- 工具类

└── config.json --- 全局配置文件

```

亮点1：routes –> index.js -> RouteItem -> RouteItem -> Route > render = DynamicCmp （动态引入import模块路由index.js实现模块化），所以后续功能功能只需在routes文件夹下添加相应的容器组件调用显示组件就行；

亮点2：config.json配置项文件提供页面展示、逻辑功能所需的数据，可根据需求自行配置，配合上react基于webpack的热加载功能，这里可以有很大的想象空间。

建议：Config.json文件太多，不利于统一管理，且在实现可配置过程中增加了很多工作量，（当然，这种设计符合模块化的思想，如何取舍可再讨论），一个比较好的方案是只在主目录下保留config.json，但是这个config.json中按模块收纳进其他模块的配置项，这只是想到的一个可替代方案。

框架可交互实现思路：主要基于前述两个亮点，即：1.动态引入模块（路由）；2.配置项的使用以及热加载。目前使用node操作config.json测试是可行的，对于深度嵌套的对象值的修改测试通过，但是有些对象中包含数组，目前还未做相应处理（建议：config.json配置项嵌套深度不要太深），但是思路是可行的。至于实现可交互具体做法，可以另建一个系统，提供基础布局以供选择，页面展示使用input（自行输入配置项的值-面向开发人员友好）、select（封装配置参数）选择、组件图形（封装配置参数）拖拽、自定义，点击保存后生成对应的config.json，框架重启。