### React 脚手架搭建全过程

###### 安装 create-react-app

npx create-react-app react-mycli

cd react-mycli

yarn start (安装yarn 命令 : npm install -g yarn )

###### 安装 Ant Desigin

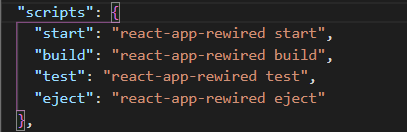
yarn add antd

###### 配置 Ant Design 按需加载

第一步：安装 react-app-rewired 为项目添加 webpack 配置，安装完成后 在项目的根目录中新建 config-overrides.js 文件；

yarn add react-app-rewired

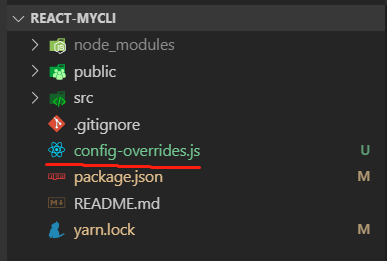
第二步：修改 package.json 文件，修改如下

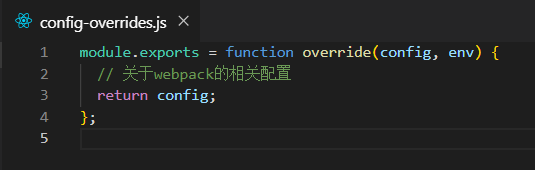


第三步：由于react-app-rewired 2.X 以后不在支持 injectBabelPlugin 的方式，需要安装 customize-cra

yarn add customize-cra

第四部：在根目录里添加 config-override.js 文件，如下所示：



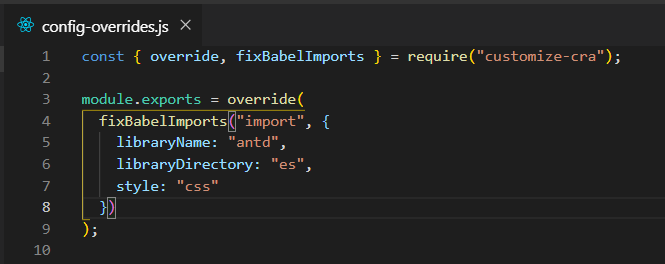


好了，基本的操作就是这样，下面我们再具体到项目中做一些配置

第五步：配置按需加载，我们需要使用 babel-plugin-import，它是一个用于按需加载组件代码和样式的babel插件

yarn add babel-plugin-import

接着配置webpack ，修改 config-overrides.js ，修改后如下所示



这样我们在组件中就可以按需引入组件库中的所需要的组件了

###### 路由配置

第一步：安装 react-router-dom react-router-config

yarn add react-router-dom

yarn add react-router-config

安装 @loadable/component 实现路由组件按需加载

yarn add @loadable/component

安装完成，一切准备就绪，下面我们就开始配置路由

第二步：在src文件夹下新建一个路由文件夹，用于放置路由的配置文件，命名 router ；在 router 里新建一个 router.js

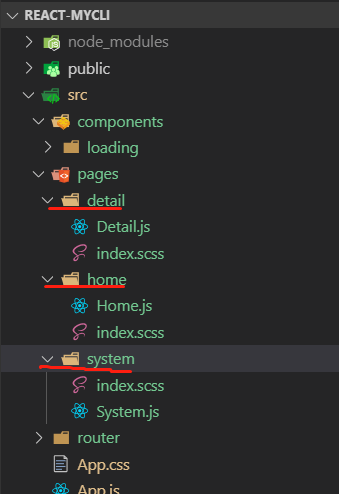
引入所需要使用的依赖包

import React from "react";

// loadable 用来实现路由组件的按需加载

import loadable from "@loadable/component";

创建组件并引入，如下所示：

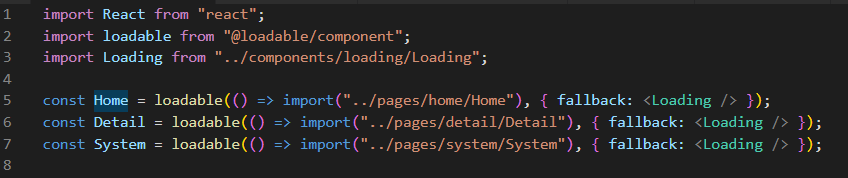


const Home = loadable(() => import("../pages/home/Home"));

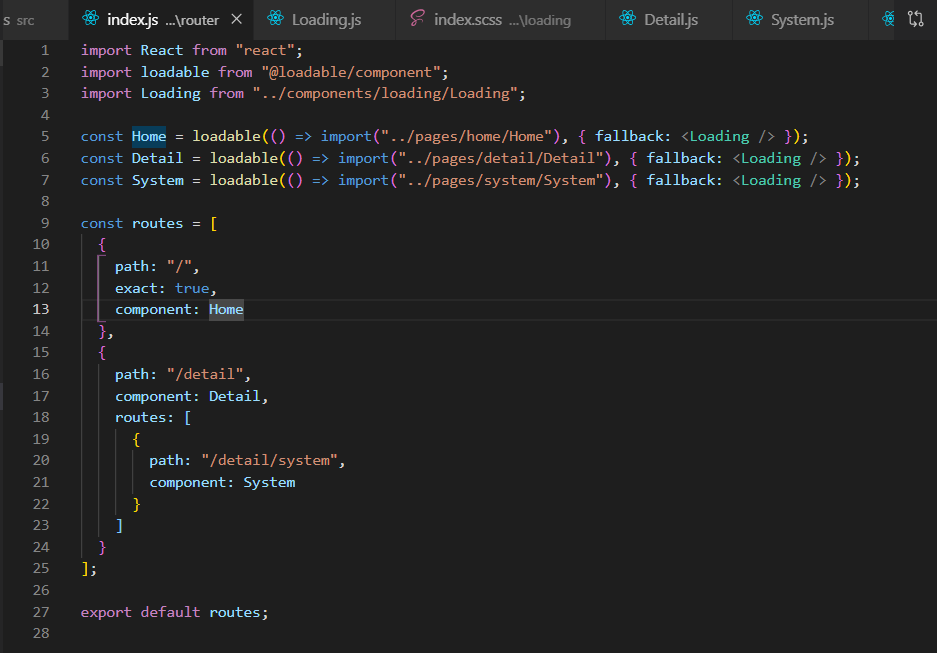
const Detail = loadable(() => import("../pages/detail/Detail"));

const System = loadable(() => import("../pages/system/System"));

创建Loading 组件，引入，用于组件加载前loading显示，增加用户体验，如下所示，至于为什么这样使用，请看 [@loadable/component 文档](https://www.npmjs.com/package/@loadable/component)



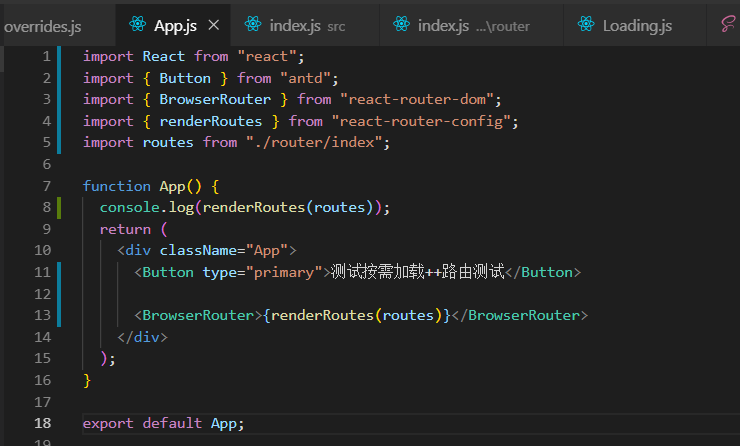
接下来就来配置路由数组吧，最终router/index.js 配置内容如下所示：



到这一步路由基本上配置完成了，下面来看看如何使用吧

首先进入 App.js 引入 react-router-dom react-router-config

其次引入路由配置文件 router/index.js 具体实现代码如下所示：

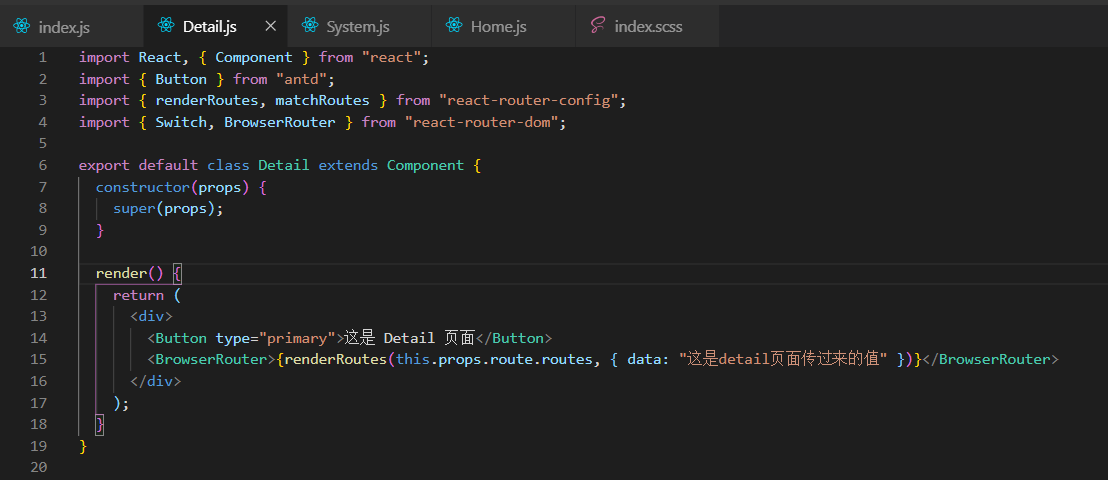


renderRoutes 是react-router-config 提供的api ，具体使用方法请参考 [react-router-config 文档](https://www.npmjs.com/package/react-router-config)

BrowserRouter 是 react-router-dom 提供的 api ，具体使用方法请参考[react-router-dom 文档](https://github.com/ReactTraining/react-router/tree/master/packages/react-router-dom)

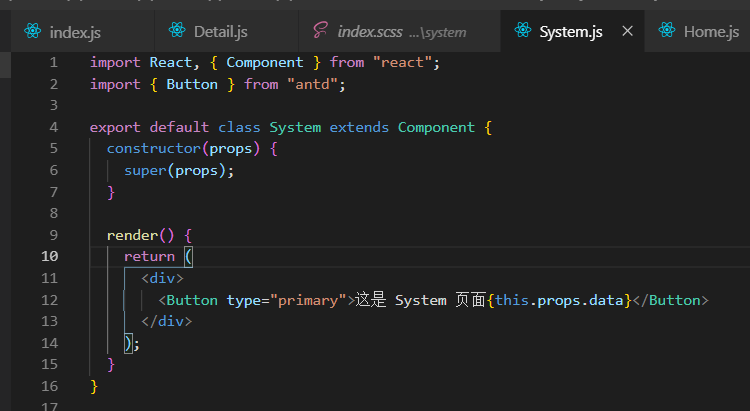
到这一步，第一层理由已经可以实现了，浏览器里输入对应的路由，即可显示对应的组件；下面来说说嵌套路由（子路由）如何实现：

从上面路由配置文件 router/index.js 文件中可以看出，/detail 下面存在子路由 /detail/system 要想实现其实很简单，只需要在Detail 组件中重新调用一下 renderRoutes 方法即可，当然在调用之前，还是需要取引用一下 react-router-config 和 react-router-dom ,子路由的获取，通过 this.props.route 获取，详细用法参考以下代码：



renderRoutes 第二个参数是一个对象，作用是向子组件中传值用，

子组件获取穿过来值得方法，只需要通过 this.props.属性 即可获取；代码实现如下：



到这里，路由的配置就已经完成了。

###### redux react-redux 集成

[react-redux](http://cn.redux.js.org/docs/react-redux/)是Redux 的作者封装了一个 React 专用的库   
[redux](https://cn.redux.js.org/)是用于react集中管理状态

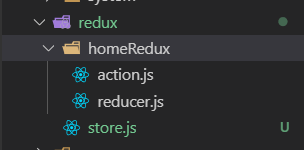
第一步：安装redux react-redux

yarn add redux

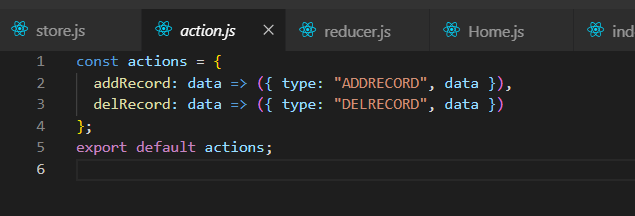
yarn add react-redux

第二步：在src 目录下创建 redux 文件夹，用于集中管理项目中的redux

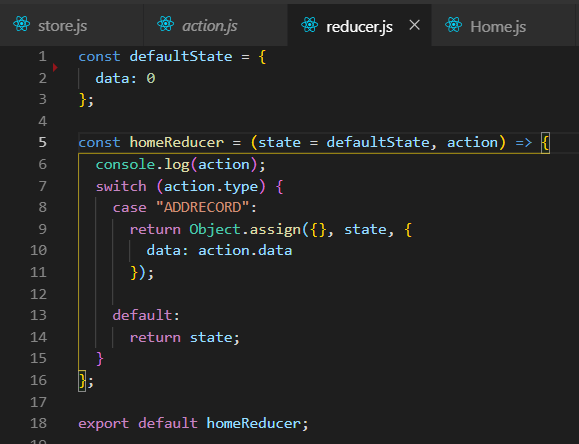
紧接着在redux目录下创建 store.js，为Home 组件创建 homeReducer 文件夹，homeRedux文件夹下创建 action.js 和 reducer.js ，目录结 构如下所示 ：



创建 action 如下所示：



创建 reducer 如下所示：

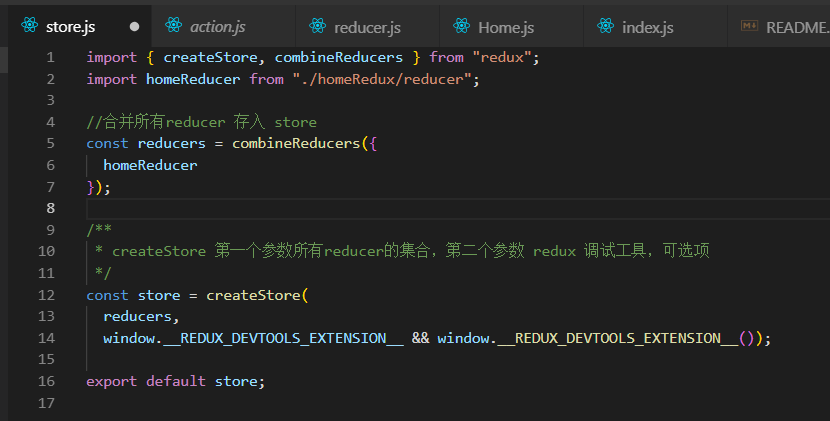


把创建好的reducer 集成进 store，在 store.js 中首先创建 store

引入 redux

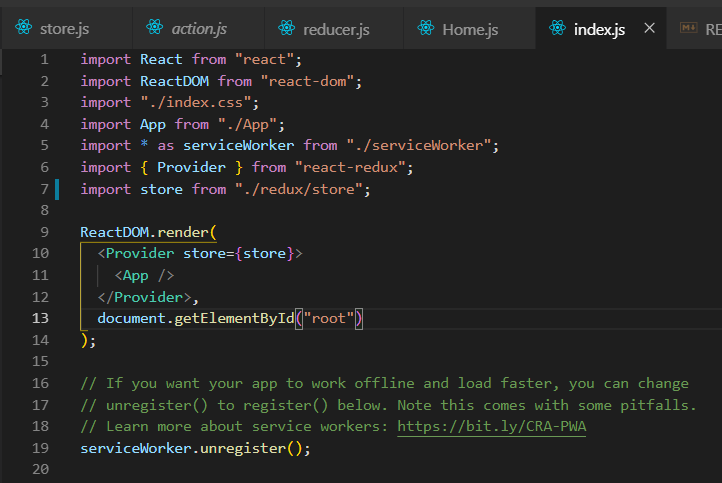
Import { createStore , combineReducer } from “ redux ”

createStore 用于创建 store ， combineReducer 用于合并所有的 reducer 存入 store ，代码如下所示：



第三步：项目中集成

要把 redux 集成到react 中 ，就需要使用到 react-redux ，react-redux 提供了 Provider 组件，在入口文件 index.js 文件中引入Provider组件，将它包在最外层，把 store 作为 prop 传入 Provider



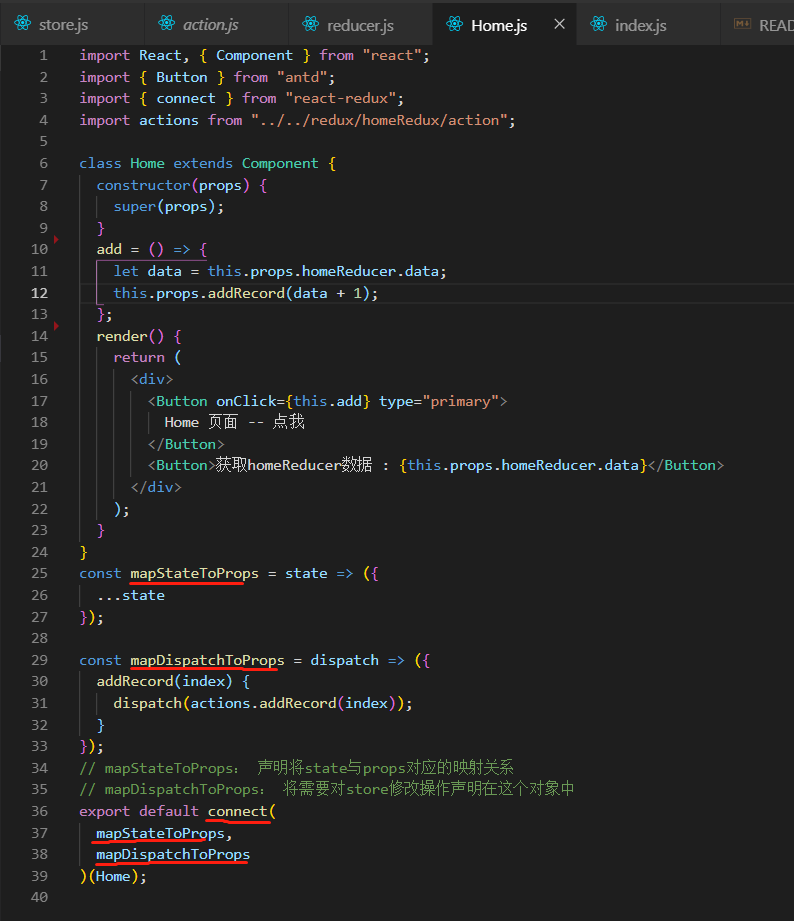
在应用组件中通过 connect 将组件与 store 关联起来，然后通过直接访问 props的方式访问 store

组件中引入 connect 以及引入对应的 action

Import { connect } from “ react-redux ”

Import actions from “ ../../redux/homeRedux/action ”

具体组件中代码如下所示 ：



其中 connect 是一个高阶函数 ，第一个参数 mapStateToProps 声明将 state 与 props 对应的映射关系，第二个参数 mapDispatchToProps 将需要对 store 修改操作声明在这个对象中

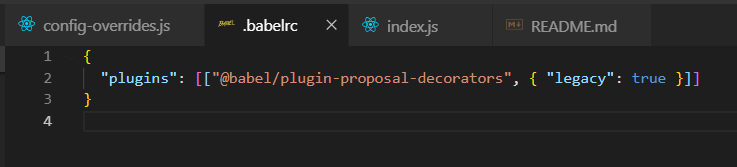
至此 redux react-redux 已经集成到项目中了，具体代码请参考 github 上代码切换到 redux-01 分支

###### 装饰器配置

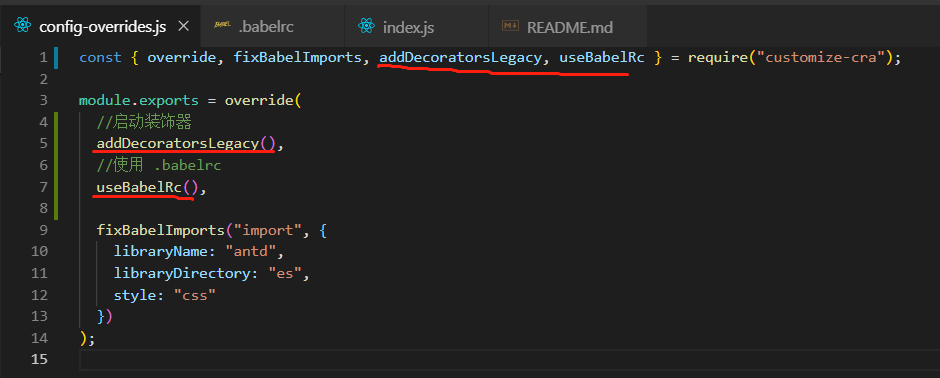
项目中使用装饰器，需要安装 @babel/plugin-proposal-decorators

yarn add @babel/plugin-proposal-decorators

在项目的根目录下添加 .babelrc 文件，.babelrc文件添加以下代码

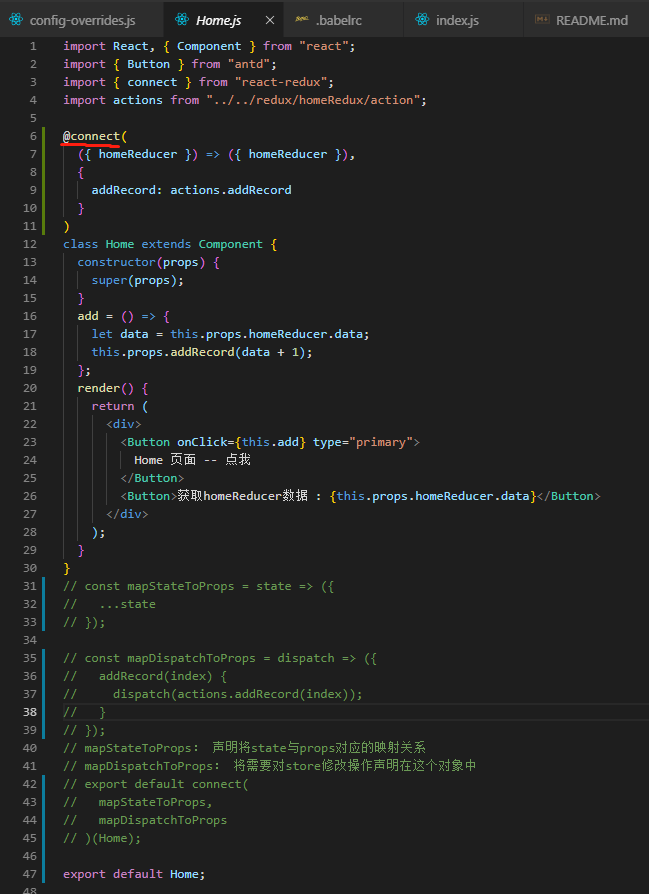


修改 config-overrides.js 以启动装饰器 ：



addDecoratorsLegacy 和 useBabelRc 是 customize-cra 提供的api，用于启动装饰器和使用 .babelrc 文件配置

使用：组件中通过 connect 连接 store 操作过于繁琐，可以使用 @connect 来简化操作，具体使用如下所示 ：



至此装饰器配置以及使用已经完成，详细代码请参考 github 上代码切换到 decorators

###### Http 服务类的封装 （用于请求后端接口）

http 请求的封装是基于 axios 的二次封装，这里就不做过多的讲解，详细请参考[axios](https://www.kancloud.cn/yunye/axios/234845) 文档 ，具体封装代码，请参考 /src/service 文件

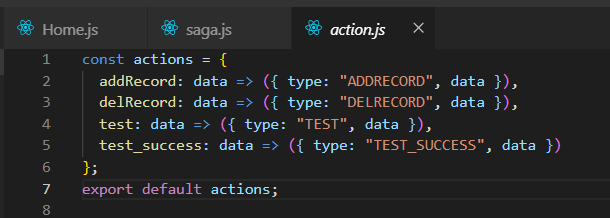
###### Redux-saga 的集成和使用

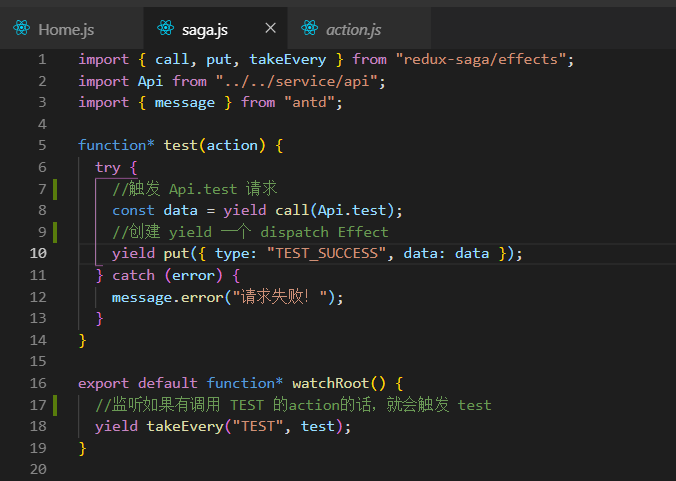
redux-saga 使用了 ES6 的 Generator 功能，让异步的流程更易于读取，写入和测试，redux-saga 相当于在 redux 原来的数据流中多了一层，对action 进行监听，接收到action时，派发一个任务维护state

第一步：安装

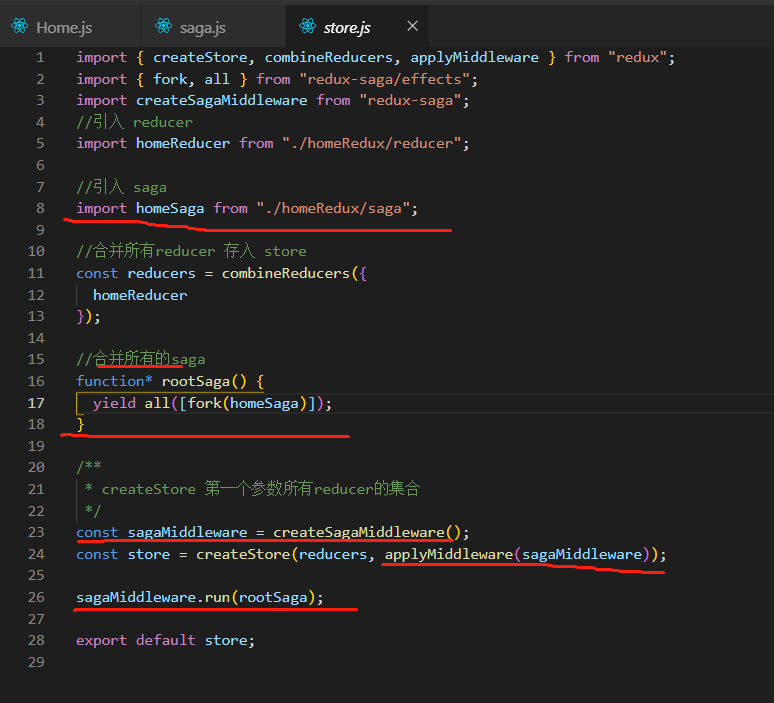
yarn add redux-saga

第二步：在homeRedux/action.js中创建action，分别是TEST 和 TEST\_SUCCESS，在homeRedux 文件夹下面创建 saga.js ，action.js 和 saga.js 代码如下所示：

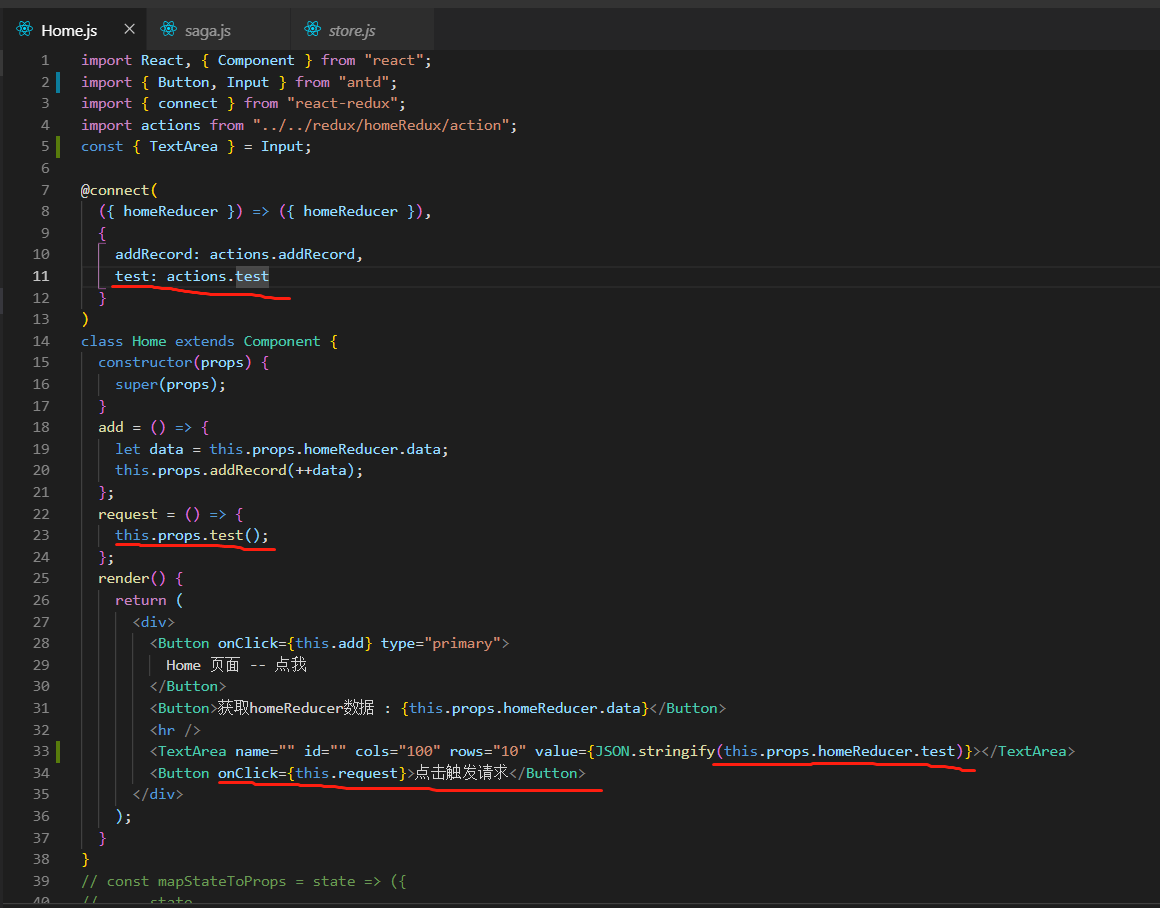




Redux-saga 详细使用请参考 [Redux-saga 文档](https://redux-saga-in-chinese.js.org/)



store.js 中进行如上配置，此时 saga 配置完成，在组件中通过触发 action 来发起请求



###### Redux-actions 的集成和使用

安装：yarn add redux-actions

Redux-actions 主要提供了 createActions 和 handleActions 来简化redux 相关代码的工具 ，具体参考代码可看 action.js 和 reducer.js

需要注意的是 ： reducer 中方法取 action 时，参数 redux-actions ，自动加了 payload 参数

###### 