从零开始搭建react项目框架

零：写在前面

绿色表示要执行的命令

红色表示要注意的内容

1. 环境

node：v10.13.0

npm：6.4.1

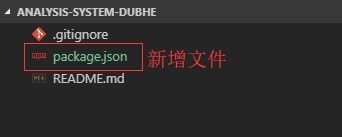
模块打包工具：parcel

技术栈：react+react-router4+redux+axios+mockjs

1. 搭建步骤
2. 创建一个文件夹(analysis-system-Dubhe)存放该项目，进入该文件夹，创建一个npm项目

npm init

npm init会问你一连串的问题，全部回车用默认选项即可，完成后项目结构如下图所示



1. 添加React、Babel和Parcel的依赖

npm install --save react

npm install --save react-dom

npm install –save-dev babel-core

npm install --save-dev babel-preset-react

npm install --save-dev babel-preset-env

npm install --save-dev parcel-bundler

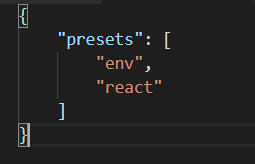
tips：

babel-core 核心包，如果某些代码需要主动调用Babel的API进行转码（基本没这种需求），可以使用babel-core模块。

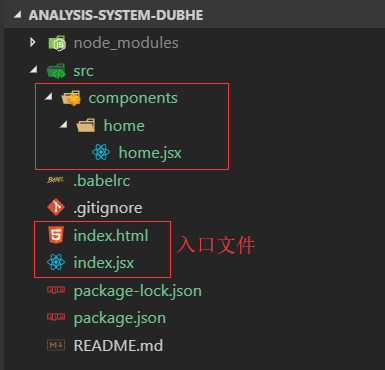
babel-preset-env 解析es的包,智能识别当前运行环境并进行转换

babel-preset-react 解析jsx的包

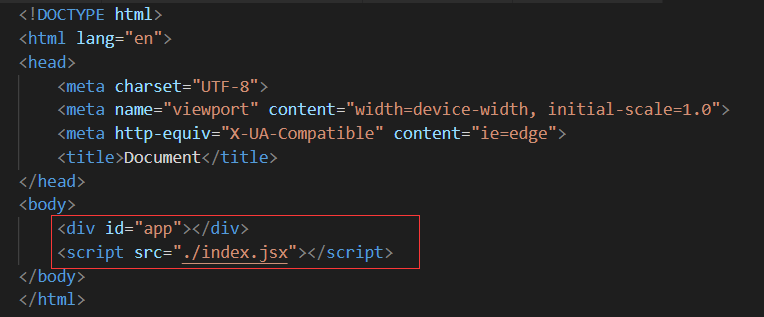
1. 创建.babelrc文件，这个文件告诉Parcel我们使用了ES6和React JSX



1. 创建React项目，创建三个文件index.html，index.jsx和home.jsx并指定目录结构如下 。



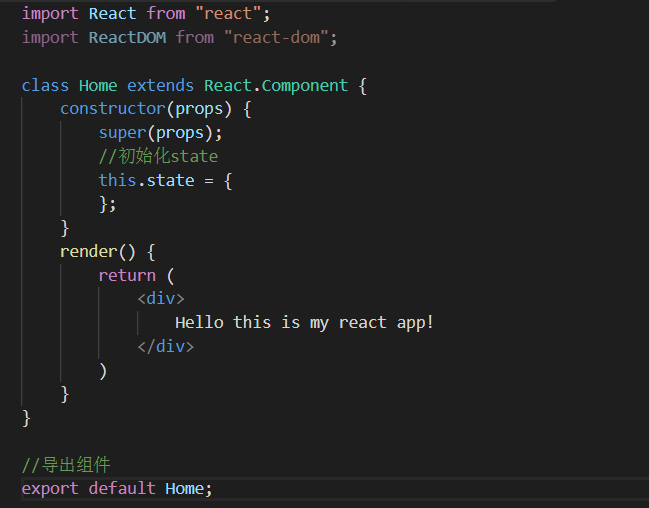
index.html文件如下



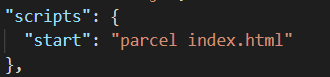
index.jsx文件如下：



home.jsx文件如下：



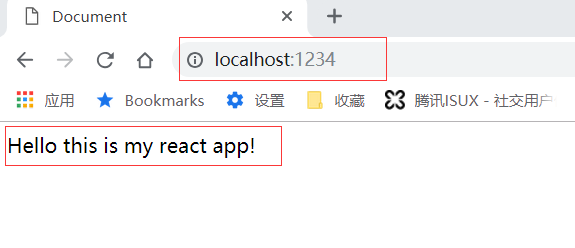
1. 在package.json中添加script脚本，用于启动我们的应用



1. 启动应用

npm run start

之后在浏览器中输入<http://localhost:1234>查看项目启动后的效果如下



1. 引入sass编写样式文件。

sass的样式文件格式为.scss，.scss的编译需要用到node-sass模块，我们用npm来安装该模块

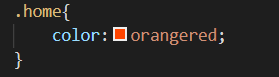
npm install node-sass --save

一旦node-sass安装完成，我们就可以在JavaScript文件中导入scss文件了。导入方式如下：

import ‘./home.scss’

接上面的例子，我们添加一个home.scss文件，并将这个文件导入到home.jsx中进行使用。

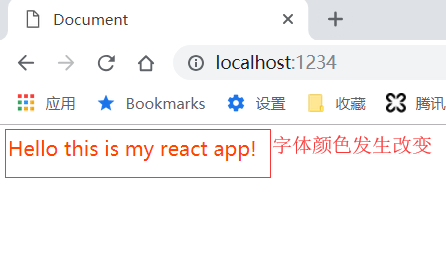
home.scss文件如下:



修改home.jsx文件如下：



重新启动应用，查看页面效果如下：



1. 引入react-router4进行路由管理

安装react-router-dom

npm install react-router-dom --save

tips：

react-router React Router 核心

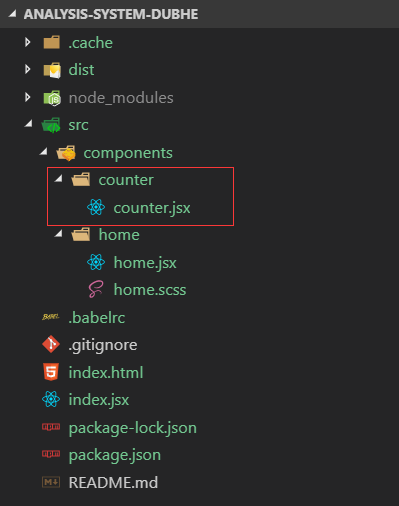
react-router-dom 用于 DOM 绑定的 React Router（基于浏览器环境开发的只需要安装这个）

react-router-native 用于 React Native 的 React Router（基于react-native环境开发的，只需要安装这个）

react-router-dom依赖react-router，所以我们使用npm安装依赖的时候，只需要安装相应环境下的库即可，不用再显式安装react-router。

接着上面的例子现在我们已经有了一个home页面，我们再添加一个counter页面，实现点击连接进行两个页面的切换的功能。

* 添加counter文件夹和counter.jsx，修改后的目录结构如下



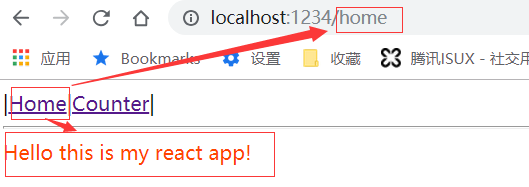
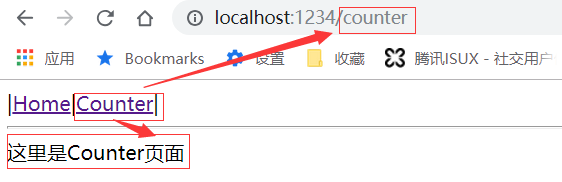
* counter.jsx内容如下



* 修改index.jsx页面



* 重新启动应用，查看页面效果如下：

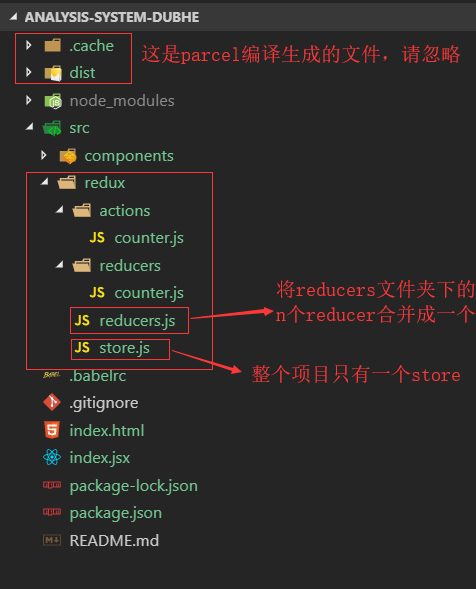
1. 引入redux进行状态管理

接着上面的代码我们在counter.jsx里做一个计数器，实现自增，自减，重置的功能。

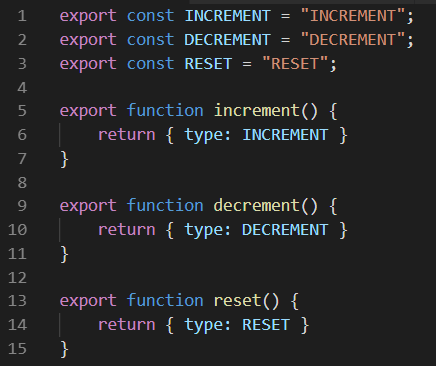
先安装redux

npm install redux –save

* 创建redux文件夹来管理我们的actions、reducers和store目录结构如下：

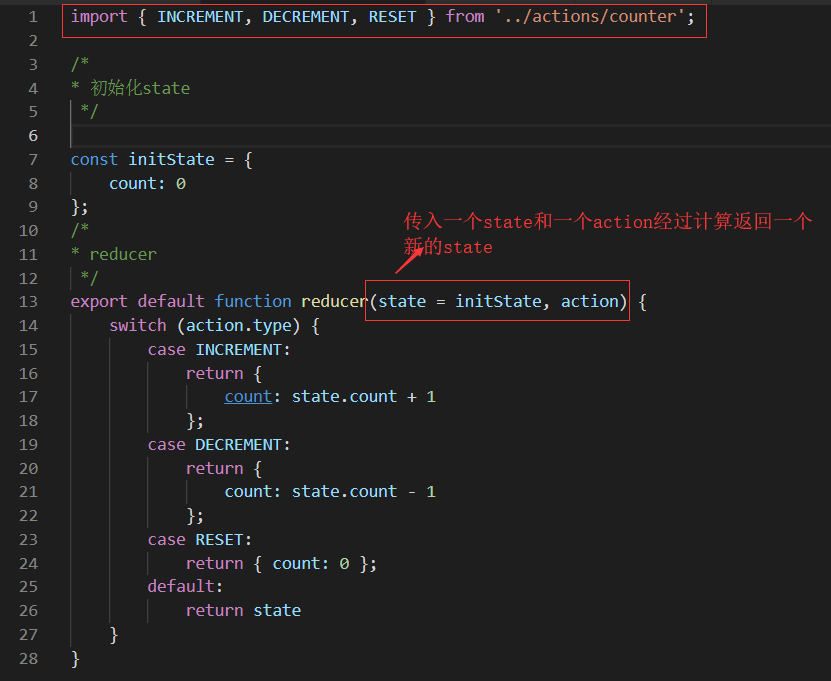


* actions/counter.js内容如下：



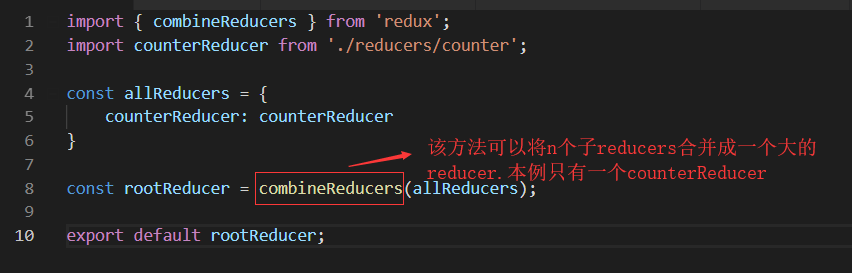
action返回的其实就是一个json对象。

* reducers/counter.js内容如下：

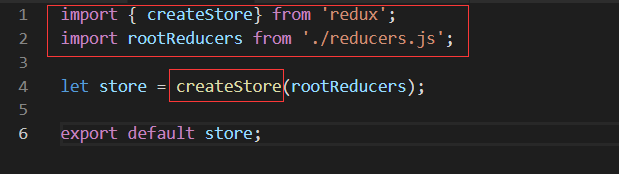


reducer就是传入一个老的state和一个action返回一个新的state的处理函数。

* reducers.js内容如下 ：

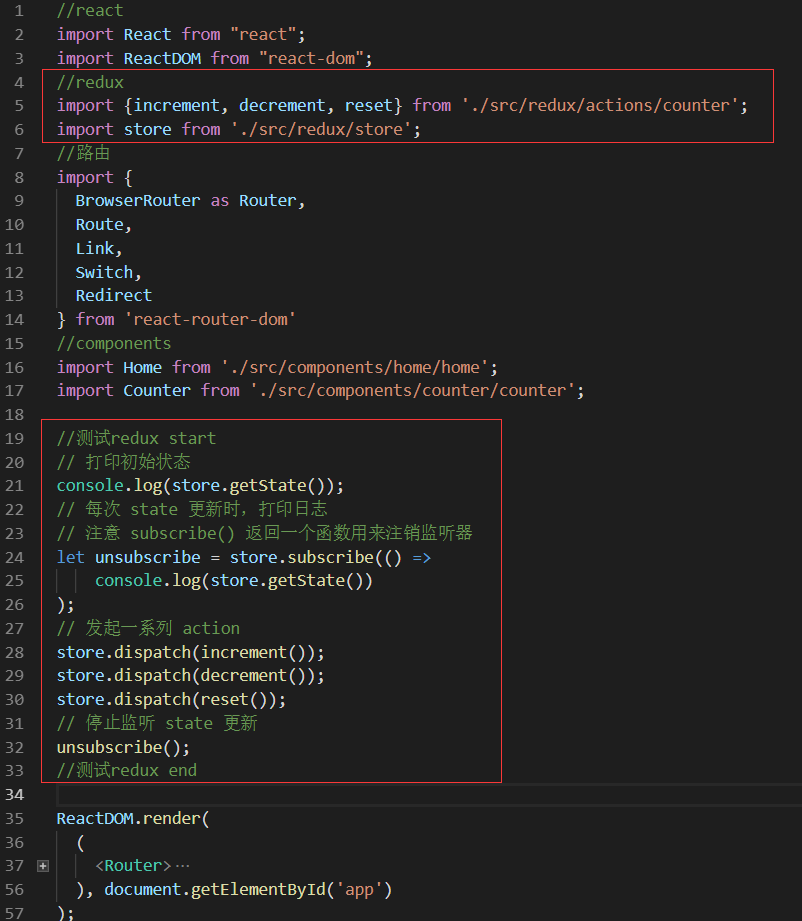


* store.js内容如下：

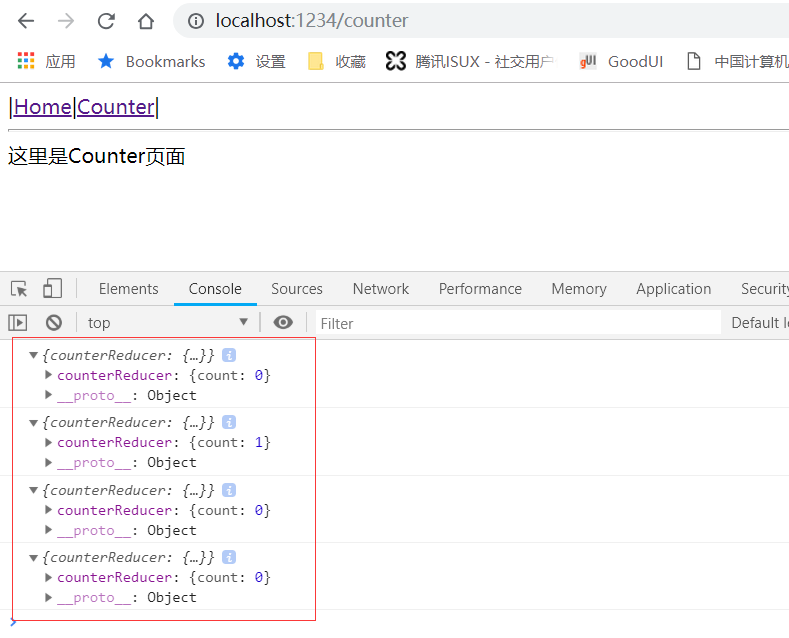


* 给index.jsx添加几行代码，我们进行一下测试。

index.jsx内容如下：



重新启动应用，查看页面效果如下：



store主要有如下作用

维持应用的 state；

提供 getState() 方法获取 state；

提供 dispatch(action) 触发reducers方法更新 state；

通过subscribe(listener) 注册监听器;

通过 subscribe(listener) 返回的函数注销监听器。

1. 开始将react和redux进行搭配使用, 使counter.jsx组件能获得到Redux的state，并且能发射action.

tips1:第9步中我们是利用手动监听，手动引入store的方法进行测试很麻烦。

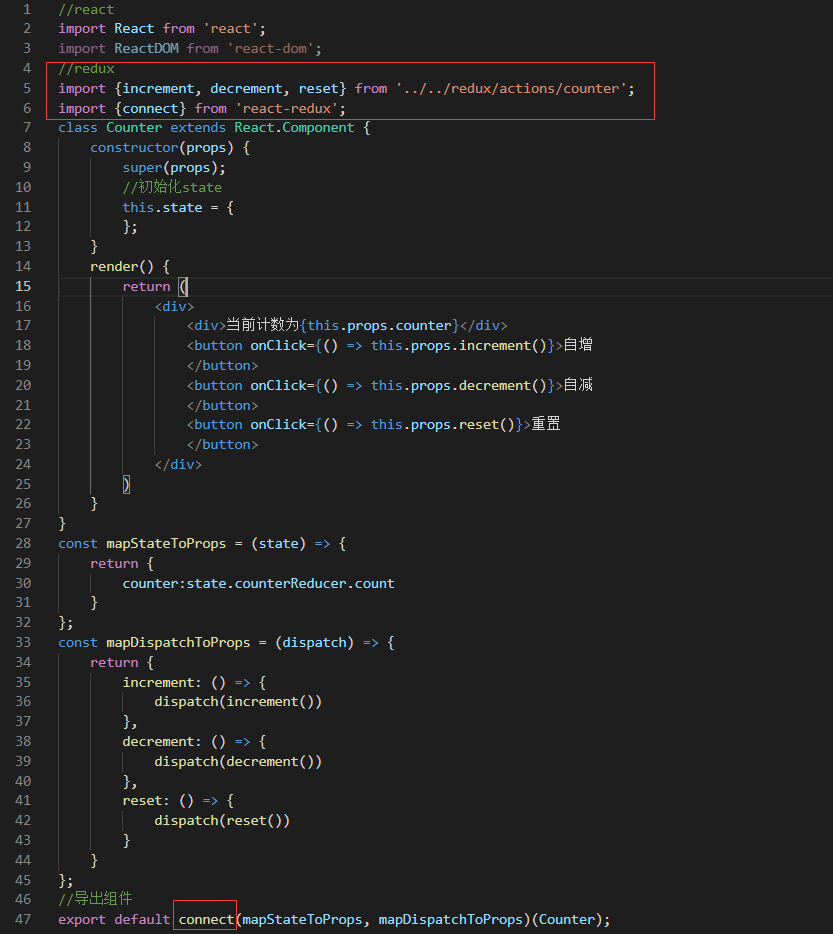
tips2:容器组件就是使用 store.subscribe() 从 Redux state 树中读取部分数据，并通过 props 来把这些数据提供给要渲染的组件。你可以手工来开发容器组件，但建议使用 React Redux 库的 connect() 方法来生成，这个方法做了性能优化来避免很多不必要的重复渲染。

tips3:react-redux提供了一个方法connect。该方法接收两个参数，一个mapStateToProps,就是把redux的state，转为组件的Props，还有一个参数是mapDispatchToprops, 就是把发射actions的方法，转为Props属性函数。(敲黑板：这段话要结合代码好好去理解)

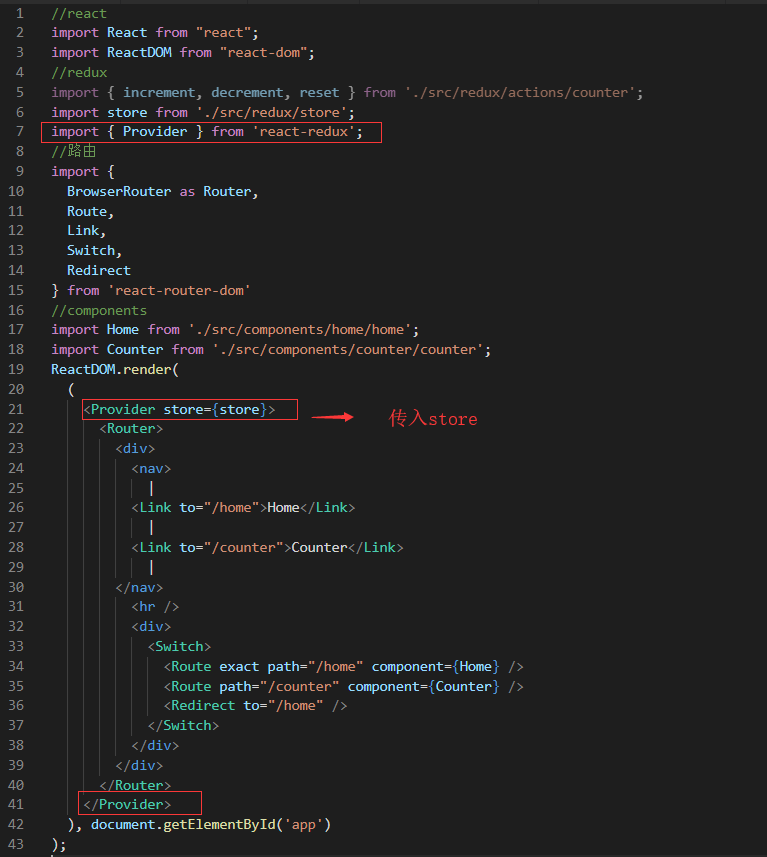
* 先安装react-redux

npm install react-redux –save

* 改写counter.jsx页面，counter.jsx内容如下：



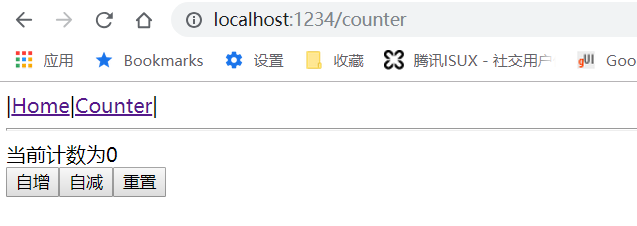
* 修改index.jsx，将我们的store传入，修改后的内容如下：



tips: 所有容器组件都可以访问 Redux store，所以可以手动监听它。一种方式是把它以 props 的形式传入到所有容器组件中。但这太麻烦了，因为必须要用 store 把展示组件包裹一层，仅仅是因为恰好在组件树中渲染了一个容器组件。

建议的方式是使用指定的 React Redux 组件Porvider来魔法般的让所有容器组件都可以访问store，而不必显示地传递它。只需要在渲染根组件时使用即可。

* 重新启动应用，查看页面效果如下,点击按钮可以实现自增，自减，重置。



到这里就完成了react和redux的结合使用总结一下这一步的知识点

Provider组件是让所有的组件可以访问到store。不用手动去传。也不用手动去监听。

connect函数作用是从 Redux state 树中读取部分数据，并通过 props 来把这些数据提供给要渲染的组件。也传递dispatch(action)函数到props。

1. 异步action的引入

Action 发出以后，Reducer 立即算出 State，这叫做同步；Action 发出以后，过一段时间再执行 Reducer，这就是异步。我们前端向后端请求数据大多数都是异步的。

要想实现异步我们需要用到applyMiddlewares()引入中间件并且要用到redux-thunk中间件来辅助实现异步功能。

tips1:applyMiddlewares方法是Redux的原生方法，作用是将所有的中间件组成一个数组，依次执行下去。所以中间件有次序要求。

tips2:redux-thunk中间件用来改造store.dispatch，使得store.dispatch可以接受函数作为参数。

tips3:因此我们第一种异步解决方案就是：写出一个返回函数的 Action Creator，然后使用redux-thunk中间件，改造store.dispatch。

* 安装redux-thunk中间件

npm install redux-thunk –save

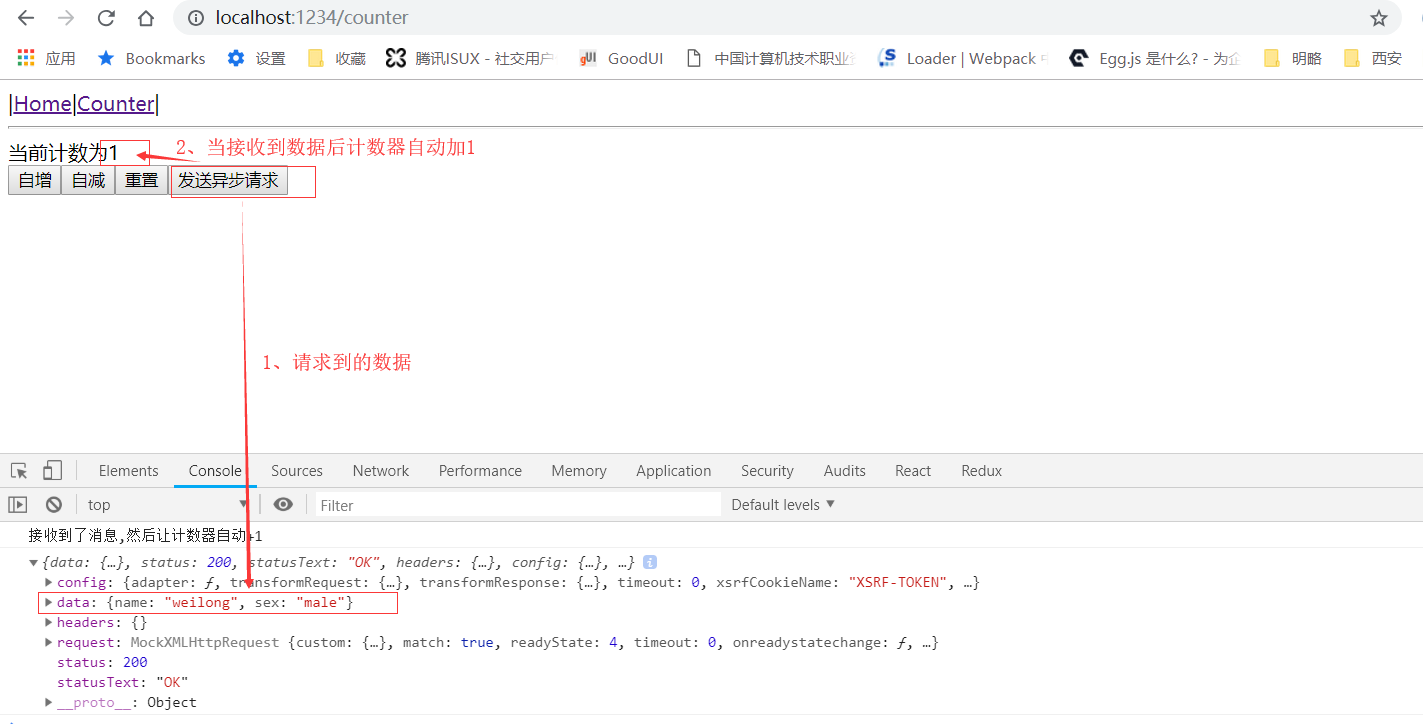
* 安装mockjs用来模拟后来数据

npm install mockjs –save-dev

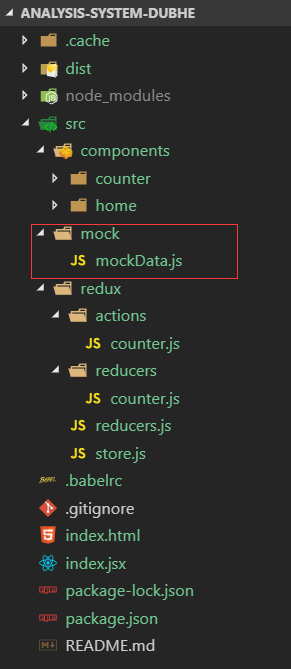
* 安装axios来进行http请求发送

npm install axios –save

* 再接着上面的例子我们在counter页面里再加一个异步请求的按钮，每点击一下按钮我们先请求一组数据，当数据请求到后我们再让计数器自增1.最终的效果如 下图所示：



* 先创建一个mock目录，里面创建一个mockData.js来当然我们http请求的数据源，目录结构如下：



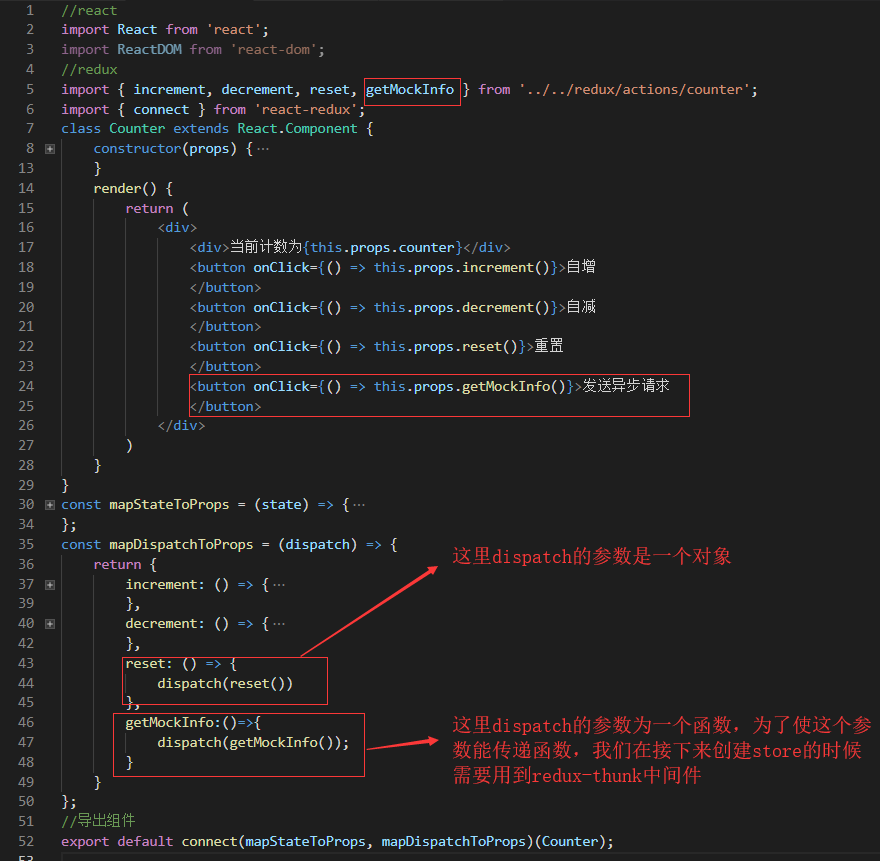
mockData.js内容如下：



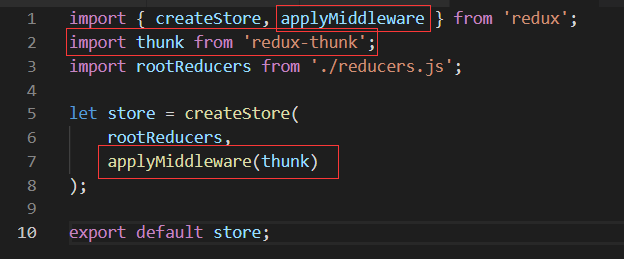
* 修改actions/counter.js添加一个异步请求，counter.js内容如下：



* 修改counter.jsx页面添加异步请求按钮。counter.jsx内容如下：



* 修改store.js



* 重启查看效果。

整个框架的搭建到这里就结束了，如果有什么常用的组件我们会再后继不断加入，比如我们添加了第12点。

1. 利用prop-types第三方库对组件的props中的变量进行类型检测

npm install prop-types –save

用法非常简单，在需要用到的时候自行百度一下就好，大概用法如下所示

