**React** 项目**-**后台管理系统

##### 尚硅谷前端研究院

第 **1** 章**:** 准备

# 项目描述

###### 此项目为一个前后台分离的后台管理的 SPA, 包括前端 PC 应用和后端应用

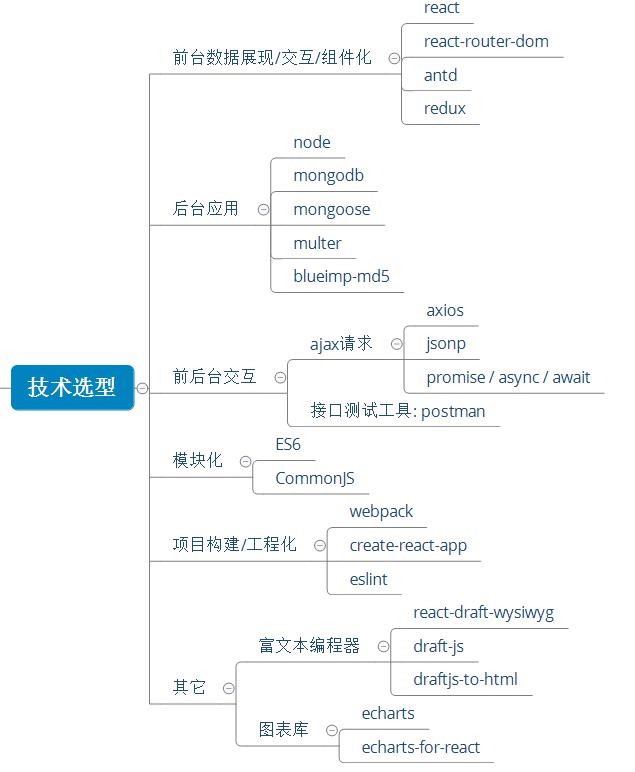
1. 包括用户管理 / 商品分类管理 / 商品管理 / 权限管理等功能模块
2. 前端: 使用 React 全家桶 + Antd + Axios + ES6 + Webpack 等技术
3. 后端: 使用 Node + Express + Mongodb 等技术
4. 采用模块化、组件化、工程化的模式开发
   1. 项目功能界面



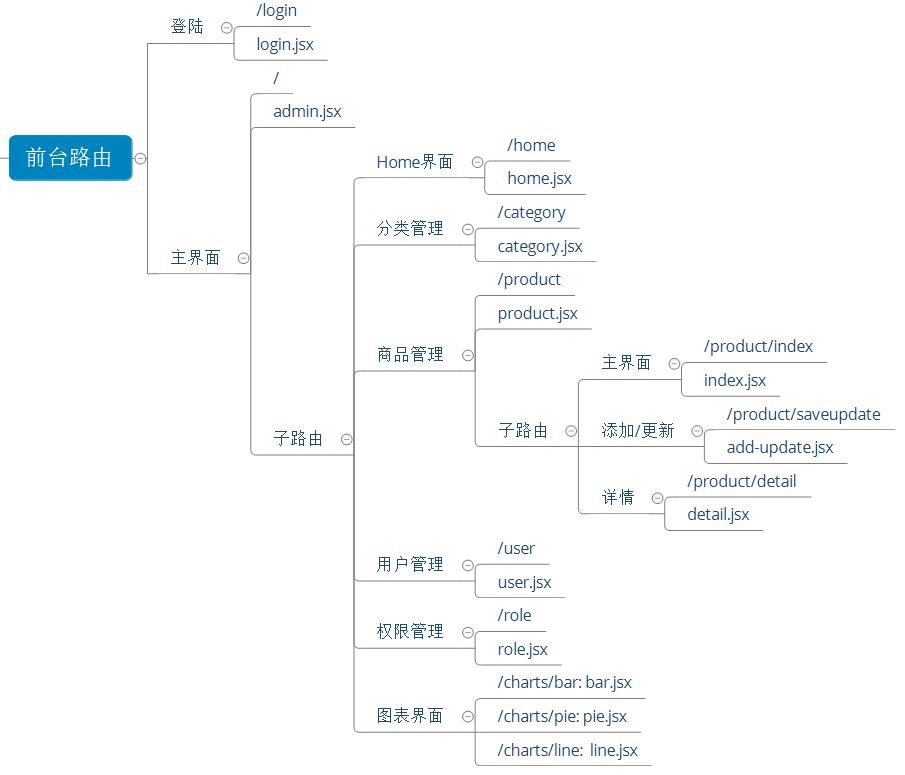


其它的界面功能运行 final 版应用查看

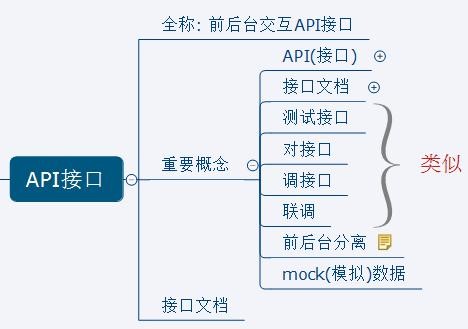
* 1. 技术选型



* 1. 前端路由



* 1. **API/**接口



# 你能从此项目中学到什么**?**

### 流程及开发方法

###### 熟悉一个项目的开发流程

1. 学会模块化、组件化、工程化的开发模式
2. 掌握使用 **create-react-app** 脚手架初始化 react 项目开发
3. 学会使用 **node+express+mongoose+mongodb** 搭建后台应用
   * 1. **React** 插件或第三方库
4. 掌握使用 **react-router-dom** 开发单页应用
5. 学会使用 **redux+react-redux+redux-thunk** 管理应用组件状态
6. 掌握使用 **axios/jsonp** 与后端进行数据交互
7. 掌握使用 **antd** 组件库构建界面
8. 学会使用 **echarts/bizcharts** 实现数据可视化展现
9. 学会使用 **react-draft-wysiwyg** 实现富文本编辑器
   1. **npm/yarn** 常用命令

yarn 命令文档: <https://yarnpkg.com/zh-Hans/docs/cli/>

npm 命令文档: <https://docs.npmjs.com/cli-documentation/>

## 设置淘宝镜像

npm config set registry https://registry.npm.taobao.org yarn config set registry https://registry.npm.taobao.org

## 初始化项目:

yarn init -y npm init -y

## 下载项目的所有声明的依赖: yarn

npm install

## 下载指定的运行时依赖包:

yarn add [webpack@3.2.1](mailto:webpack@3.2.1)

npm install [webpack@3.2.1](mailto:webpack@3.2.1) -S

## 下载指定的开发时依赖:

yarn add [webpack@3.2.1](mailto:webpack@3.2.1) -D npm install [webpack@3.2.1](mailto:webpack@3.2.1) -D

## 全局下载指定包:

yarn global add webpack npm install webpack -g

## 删除依赖包:

yarn remove webpack npm remove webpack -S

yarn global remove webpack npm remove webpack -g

## 运行项目中配置的 script: yarn run xxx

npm run xxx

## 查看某个包的信息:

yarn info xxx npm info xxx

* 1. **git** 常用基本命令

Git 在线参考手册: <http://gitref.justjavac.com/>

* git config --global user.name "username" //配置用户名
* git config --global user.email "[xx@gmail.com](mailto:xx@gmail.com)" //配置邮箱
* git init //初始化生成一个本地仓库
* git add . //添加到暂存区
* git commit –m "message" //提交到本地仓库
* git remote add origin url //关联到远程仓库
* git push origin master //推送本地 master 分支到远程 master 分支
* git checkout -b dev //创建一个开发分支并切换到新分支
* git push ogigin dev //推送本地 dev 分支到远程 dev 分支
* git pull origin dev //从远程 dev 分支拉取到本地 dev 分支
* git clone url //将远程仓库克隆下载到本地

\* git checkout -b dev origin/dev // 克隆仓库后切换到 dev 分支

第 **2** 章**:** 应用开发详解

# 开启项目开发

* + 1. 使用 **create-react-app(**脚手架**)**搭建项目

###### create-react-app 是 react 官方提供的用于搭建基于 react+webpack+es6 项目的脚手架

1. 操作:

npm install -g create-react-app : 全局下载工具

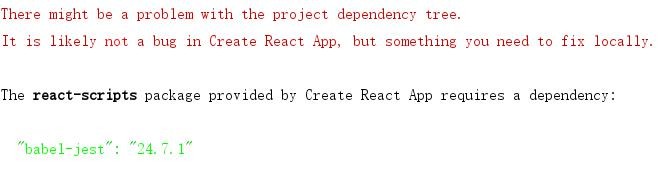
create-react-app react-admin :下载模板项目

cd react-admin

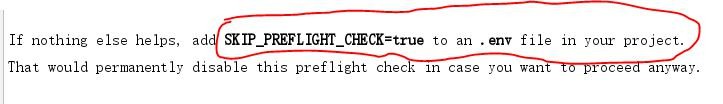
npm start

访问: localhost:3000

1. 部分同学可能会出现包版本差异的导致异常的问题



解决: 添加.env 配置文件忽略版本差异



SKIP\_PREFLIGHT\_CHECK=true

* + 1. 编码测试与打包发布项目

1. 编码测试

npm start

访问: http://localhost:3000

编码, 自动编译打包刷新(live-reload), 查看效果

1. 打包发布

npm run build

npm install -g serve serve build

访问: http://localhost:5000

* 1. 功能需求分析

演示项目功能, 对功能模块进行分析说明

* 1. 项目源码基本目录设计
     1. 基本结构



* + 1. **App.js**

**import** React, {Component} **from 'react'**

*/\**

应用根组件

*\*/*

**class** App **extends** Component { render() {

return (

<**div**>App</**div**>

)

}

}

export default App

* + 1. **index.js**

**import** React **from 'react'**

**import** ReactDOM **from 'react-dom'**

import App from './App'

ReactDOM.render(<**App** />, **document**.getElementById(**'root'**))

* 1. 引入 **antd**

###### 参考文档:

<https://ant.design/docs/react/use-with-create-react-app-cn>

* + 1. 下载组件库包

**yarn add antd**

* + 1. 实现组件的按需打包

1. 下载依赖模块

**yarn add react-app-rewired customize-cra babel-plugin-import**

1. 定义加载配置的js 模块: config-overrides.js

**const** {*override*, *fixBabelImports*} = ***require***(**'customize-cra'**);

* 修改配置: package.json

**"scripts"**: {

"start": "react-app-rewired start", "build": "react-app-rewired build", "test": "react-app-rewired test", "eject": "react-scripts eject"

},

***module***.exports = *override*( *fixBabelImports*(**'import'**, {

**libraryName**: **'antd'**, **libraryDirectory**: **'es'**, **style**: **'css'**,

}),

);

* + 1. 在应用中使用 **antd** 组件

**import** React, {Component} **from 'react' import** {Button, message} **from 'antd'**

*/\**

应用根组件

*\*/*

**class** App **extends** Component {

*handleClick* = () => { message.success(**'**成功啦**...'**);

}

render() {

return (

<**Button type='primary' onClick=**{**this**.*handleClick*}>学习</**Button**>

)

}

}

export default App

* + 1. 自定义 **antd** 主题

###### 需求:

使 antd 的默认基本颜色从 Blue 变为 Green

下载工具包:

**yarn add less less-loader**

修改 config-overrides.js

**const** {*override*, *fixBabelImports*, *addLessLoader*} = ***require***(**'customize-cra'**);

***module***.exports = *override*( *fixBabelImports*(**'import'**, {

**libraryName**: **'antd'**, **libraryDirectory**: **'es'**, **style**: **true**,

}),

*addLessLoader*({

javascriptEnabled: true,

**modifyVars**: {**'@primary-color'**: **'#1DA57A'**},

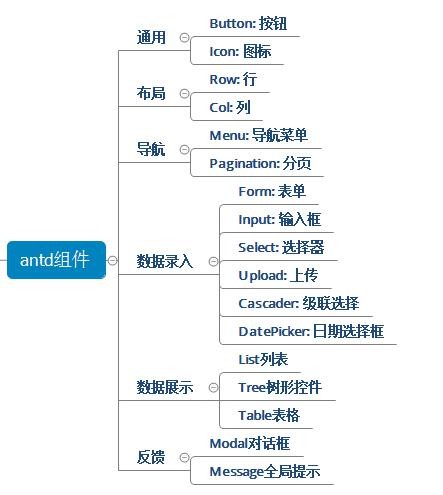
}),

);

* + 1. 应用中使用的组件



* + 1. 项目中用到的 **antd** 组件

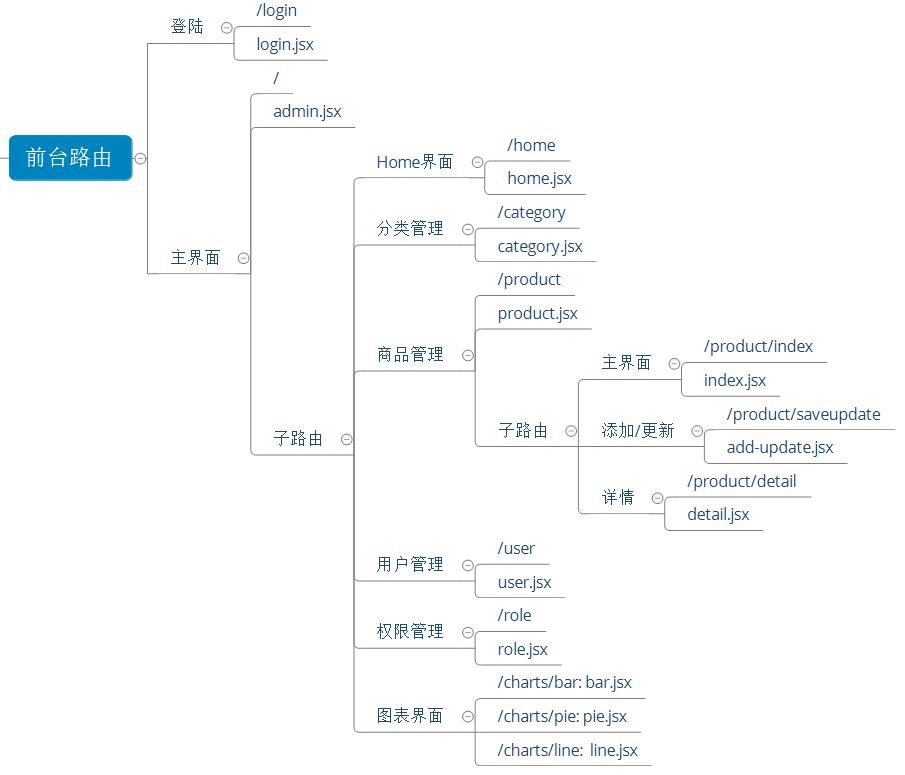


# 引入路由

* + 1. 下载路由包**: react-router-dom**

**yarn add react-router-dom**

### 前台应用路由



* + 1. 路由组件**: pages/login/login.jsx**

*/\**

用户登陆的路由组件

*\*/*

**import** React, {Component} **from 'react'**

**export default class** Login **extends** Component { render () {

return (

<**div**>login</**div**>

)

}

}

* + 1. 后台管理主路由组件**: pages/admin/admin.jsx**

*/\**

后台管理主路由组件

*\*/*

**import** React, {Component} **from 'react'**

**export default class Admin extends** Component { render() {

return (

<**div**>Admin</**div**>

)

}

}

* + 1. 映射路由**: App.js**

**import** React, {Component} **from 'react'**

import {BrowserRouter, Switch, Route} from 'react-router-dom'

import Login from './pages/login/login' import Admin from './pages/admin/admin'

*/\**

应用根组件

*\*/*

**class** App **extends** Component {

render() {

return (

<BrowserRouter>

<**Switch**>

<**Route path='/login' component=**{Login}/>

<**Route path='/' component=**{Admin}/>

</**Switch**>

</BrowserRouter>

)

}

}

export default App

* 1. **Login** 组件**(**不与后台交互**)**



### 静态组件

1. 图片资源

assets/images/logo.png assets/images/bg.jpg

1. public/css/reset.css

**html**, **body**, **p**,

**ol**,

**ul**,

**li**,

**dl**,

**dt**,

**dd**, **blockquote**, **figure**, **fieldset**,

———————————————————————————

legend,

**textarea**, **pre**, **iframe**, **hr**,

**h1**,

**h2**,

**h3**,

**h4**,

**h5**,

**h6** {

**margin**: 0;

**padding**: 0;

}

**h1**,

**h2**,

**h3**,

**h4**,

**h5**,

**h6** {

**font-size**: 100%;

**font-weight**: **normal**;

}

**ul** {

**list-style**: **none**;

}

**button**,

**input**, **select**, **textarea** {

**margin**: 0;

}

html {

box-sizing: border-box;

}

**\***, **\***:**before**, **\***:**after** {

box-sizing: inherit;

### ———————————————————————————

}

**img**, **embed**, **iframe**, **object**, **video** {

**height**: **auto**; **max-width**: 100%;

}

**audio** {

**max-width**: 100%;

}

**iframe** {

**border**: 0;

}

**table** {

**border-collapse**: **collapse**; **border-spacing**: 0;

}

**td**,

**th** {

**padding**: 0;

**text-align**: **left**;

}

**html**, **body** {

**height**: 100%;

}

**#root** {

**width**: 100%;

**height**: 100%;

}

##### 注意: 必须在 index.html 中引入

1. login/login.less

.**login** {

**width**: 100%;

**height**: 100%;

**background-image**: **url**(**'./images/bg.jpg'**); **background-size**: 100% 100%;

.login-header {

**display**: **flex**;

align-items: center;

**height**: 80**px**;

**background-color**: **rgba**(21, 20, 13, 0.5);

**img** {

**width**: 40**px**; **height**: 40**px**; **margin-left**: 50**px**;

}

**h1** {

**font-size**: 30**px**; **color**: **white**; **margin**: 0 0 0 15**px**;

}

}

.**login-content** {

**margin**: 50**px auto**; **width**: 400**px**; **height**: 300**px**;

**background-color**: **#fff**; **padding**: 20**px** 40**px**;

**h3** {

**font-size**: 30**px**; **font-weight**: **bold**; **text-align**: **center**; **margin-bottom**: 20**px**;

}

.login-form {

.login-form-button {

**width**: 100%;

}

}

}

}

### login/login.jsx

**import** React, {Component} **from 'react' import** {

Form, Input, Icon, Button,

} **from 'antd'**

**import** logo **from './images/logo.png' import './login.less'**

**const** Item = Form.*Item*

*/\**

登陆路由组件

*\*/*

**class** Login **extends** Component {

render() {

**return** (

<**div className='login'**>

<**header className='login-header'**>

<**img src=**{logo} **alt="logo"**/>

<**h1**>React 项目: 后台管理系统</**h1**>

</**header**>

<**section className='login-content'**>

<**h3**>用户登陆</**h3**>

<**Form onSubmit=**{**this**.login} **className="login-form"**>

<**Item**>

<**Input prefix=**{<**Icon type="user" style=**{{**color**: **'rgba(0,0,0,.25)'**}}/>}

**placeholder="**用户名**"**/>

</**Item**>

<**Item**>

<**Input prefix=**{<**Icon type="lock" style=**{{**color**: **'rgba(0,0,0,.25)'**}}/>}

**type="password" placeholder="**密码**"**/>

</**Item**>

<**Item**>

<Button type="primary" htmlType="submit" className="login-form-button">

登录

</**Button**>

</**Item**>

</**Form**>

</**section**>

</**div**>

)

}

}

**export default** Login

* + 1. 前台表单验证与数据收集

用户名/密码的合法性要求:

*/\**

用户名*/*密码的的合法性要求

1. 必须输入
2. 必须大于等于*4* 位
3. 必须小于等于*12* 位
4. 必须是英文、数字或下划线组成

*\*/*

**import** React, {Component} **from 'react' import** {

Form, Input, Icon, Button,

} from 'antd'

import logo from './images/logo.png' import './login.less'

**const** Item = Form.*Item*

*/\**

登陆路由组件

———————————————————————————

*\*/*

**class** Login **extends** Component {

*/\**

登陆

*\*/*

*login* = (e) => {

*//* 阻止事件默认行为*(*不提交表单*)*

e.preventDefault()

*//* 进行表单所有控件的校验

**this**.**props**.form.validateFields(**async** (err, values) => {

**if** (!err) {

*//* 校验成功

**const** {username, password} = values

***console***.log(**'**提交登陆请求**'**, username, password)

} **else** {

*//* 校验失败

***console***.log(err)

}

})

}

*/\*\**

*\** 自定义表单的校验规则

*\*/*

*validator* = (rule, value, callback) => {

*// console.log(rule, value)*

**const** length = value && value.**length const** pwdReg = /^[a-zA-Z0-9\_]+$/

**if** (!value) {

*// callback* 如果不传参代表校验成功，如果传参代表校验失败，并且会提示错误

callback(**'**必须输入密码**'**)

} **else if** (length < 4) {

callback(**'**密码必须大于 **4** 位**'**)

} **else if** (length > 12) { callback(**'**密码必须小于 **12** 位**'**)

} **else if** (!pwdReg.test(value)) { callback(**'**密码必须是英文、数组或下划线组成**'**)

} **else** {

callback() *//* 必须调用*callback*

}

### ———————————————————————————

}

render() {

**const** {getFieldDecorator} = **this**.**props**.form **return** (

<div className='login'>

<header className='login-header'>

<**img src=**{logo} **alt="logo"**/>

<**h1**>React 项目: 后台管理系统</**h1**>

</**header**>

<section className='login-content'>

<**h3**>用户登陆</**h3**>

<**Form onSubmit=**{**this**.*login*} **className="login-form"**>

<**Item**>

{

将被 *form* 接管

组成**'**}

*/\**

*getFieldDecorator* 是一个高阶函数*(*返回值是一个函数*) getFieldDecorator(*标识名称，配置对象*)(*组件标签*)* 返回新的标签

经过 *getFieldDecorator* 包装的表单控件会自动添加 *value* 和*onChange*，数据同步

*\*/*

getFieldDecorator(**'username'**, {

*//* 根据内置验证规则进行声明式验证

**rules**: [

{**required**: **true**, **whitespace**: **true**, **message**: **'**必须输入用户名**'**},

{**min**: 4, **message**: **'**用户名必须大于 **4** 位**'**},

{**max**: 12, **message**: **'**用户名必须小于 **12** 位**'**},

{**pattern**: /^[a-zA-Z0-9\_]+$/, **message**: **'**用户名必须是英文、数组或下划线

]

})(

<**Input prefix=**{<**Icon type="user" style=**{{**color**: **'rgba(0,0,0,.25)'**}}/>} **placeholder="**用户名**"**/>

)

}

</**Item**>

<**Item**>

{

getFieldDecorator(**'password'**, {

**rules**: [

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <**Input prefix=**{ | | | <**Icon type="lock" style=** | |
| **'**}} | /> | } **type="password"** | |  |

* 1. 运行 **server** 端项目

*//* 自定义表单校验规则

{**validator**: **this**.*validator*}

]

})(

{{**color**:

**'rgba(0,0,0,.25)**

**placeholder="**密码**"**/>

)

}

</**Item**>

<**Item**>

<**Button type="primary" htmlType="submit" className="login-form-button"**>

登录

</**Button**>

</**Item**>

</**Form**>

</**section**>

</**div**>

)

}

}

*/\**

用户名*/*密码的的合法性要求

1. 必须输入
2. 必须大于*4* 位
3. 必须小于*12* 位
4. 必须是英文、数组或下划线组成

*\*/*

**export default** Form.*create*()(Login)

### 说明

###### 咱们的项目是一个前后台分离的项目: 前台应用与后台应用

1. 后台应用负责处理前台应用提交的请求, 并给前台应用返回 json 数据
2. 前台应用负责展现数据, 与用户交互, 与后台应用交互
   * 1. 运行后台应用
3. 确保启动 mongodb 服务
4. 启动服务器应用: npm start
   * 1. **API** 接口文档

## 目录：

1. 登陆
2. 添加用户
3. 更新用户
4. 获取所有用户列表
5. 删除用户
6. 获取一级或某个二级分类列表
7. 添加分类
8. 更新品类名称
9. 根据分类 ID 获取分类
10. 获取商品分页列表
11. 根据 ID/Name 搜索产品分页列表
12. 添加商品
13. 更新商品
14. 对商品进行上架/下架处理
15. 上传图片
16. 删除图片
17. 添加角色
18. 获取角色列表
19. 更新角色(给角色设置权限)
20. 获取天气信息(jsonp)

**2.14.4.** 使用 **postman** 工具测试接口

###### postman 是用来测试 API 接口的工具

1. postman 可以看作活接口文档
   1. 前后台交互 **ajax**

### 下载依赖包

**yarn add axios**

* + 1. 封装 **ajax** 请求模块

###### api/ajax.js

*/\**

能发送*ajax* 请求的函数模块包装*axios*

函数的返回值是*promise* 对象 *axios.get()/post()*返回的就是*promise* 对象返回自己创建的*promise* 对象*:*

统一处理请求异常

异步返回结果数据*,* 而不是包含结果数据的 *response*

*\*/*

**import** axios **from 'axios' import** {message} **from 'antd'**

**export default function** ajax(url, data = {}, method = **'GET'**) {

**return new** Promise(**function** (resolve, reject) {

**let** promise

*//* 执行异步*ajax* 请求

**if** (method === **'GET'**) {

promise = axios.get(url, {**params**: data}) *// params* 配置指定的是*query* 参数

} **else** {

promise = axios.post(url, data)

}

promise.then(response => {

*//* 如果成功了*,* 调用*resolve(response.data)*

resolve(response.**data**)

}).catch(error => { *//* 对所有*ajax* 请求出错做统一处理*,* 外层就不用再处理错误了

*//* 如果失败了*,* 提 示 请 求 后 台 出 错

message.error(**'**请求错误**: '** + error.**message**)

})

})

}

1. api/index.js

*/\**

包含*n* 个接口请求函数的模块每个函数返回*promise*

*\*/*

**import** *ajax* **from './ajax'**

*//* 登陆

**export const** *reqLogin* = (username, password) => *ajax*(**'/login'**, {username, password},

'POST')

* + 1. 配置代理

package.json

"proxy": "http://localhost:5000"

* + 1. 请求测试**: login.jsx**

*//* 请求后台登陆

*login* = **async** (username, password) => {

**console**.log(**'**发送登陆的 **ajax** 请求**'**, username, password)

**const** result = **await** *reqLogin*(username, password)

**console**.log(**'login()'**, result)

}

* 1. **Login** 组件**(**完成登陆功能**)**

### 下载依赖

**yarn add store**

* + 1. **utils/memoryUtils.js**

*/\**

用来在内存中缓存数据的工具对象

*\*/*

export default {

**user**: {} *//* 内存中保存登陆的*user* 信息对象

}

* + 1. **login/login.jsx**

**import** React, {Component} **from 'react' import** {

Form, Icon, Input, Button, message

} from 'antd'

import './login.less'

import logo from './images/logo.png'

**import** memoryUtils **from '../../utils/memoryUtils' import** {*reqLogin*} **from '../../api'**

**const** Item = Form.*Item //* 不能写在*import* 之前

*/\**

登陆的路由组件

*\*/*

**class** Login **extends** Component {

*handleSubmit* = (event) => {

*//* 阻止事件的默认行为

event.preventDefault()

*//* 对所有表单字段进行检验

**this**.**props**.form.validateFields(**async** (err, values) => {

*//* 检验成功

**if** (!err) {

*// console.log('*提交登陆的*ajax* 请求*', values)*

**const** {username, password} = values

**const** result = **await** *reqLogin*(username, password)

*// console.log('login()', result)*

**if**(result.**status** === 0) {

*//* 提示登录成功

message.success(**'**登录成功**'**, 2)

*//* 保存用户登录信息

memoryUtils.**user** = result.**data**

*//* 跳转到主页面

**this**.**props**.history.replace(**'/'**)

} **else** {

*//* 登录失败*,* 提示错误

message.error(result.msg)

}

} **else** {

***console***.log(**'**检验失败**!'**)

}

});

*//* 得到*form* 对象

*// const form = this.props.form*

*// //* 获取表单项的输入数据

*// const values = form.getFieldsValue()*

*// console.log('handleSubmit()', values)*

}

*/\**

对密码进行自定义验证

———————————————————————————

*\*/*

*/\**

用户名*/*密码的的合法性要求

* + - 1. 必须输入
      2. 必须大于等于*4* 位
      3. 必须小于等于*12* 位
      4. 必须是英文、数字或下划线组成

*\*/*

*validatePwd* = (rule, value, callback) => {

***console***.log(**'validatePwd()'**, rule, value)

**if**(!value) {

callback(**'**密码必须输入**'**)

} **else if** (value.**length**<4) { callback(**'**密码长度不能小于 **4** 位**'**)

} **else if** (value.**length**>12) { callback(**'**密码长度不能大于 **12** 位**'**)

} **else if** (!/^[a-zA-Z0-9\_]+$/.test(value)) { callback(**'**密码必须是英文、数字或下划线组成**'**)

} **else** {

callback() *//* 验证通过

}

*// callback('xxxx') //* 验证失败*,* 并指定提示的文本

}

render () {

*//* 得到具强大功能的*form* 对象

**const** form = **this**.**props**.form

**const** { getFieldDecorator } = form;

**return** (

<**div className="login"**>

<**header className="login-header"**>

<**img src=**{logo} **alt="logo"**/>

<**h1**>React 项目: 后台管理系统</**h1**>

</**header**>

<section className="login-content">

<**h2**>用户登陆</**h2**>

<**Form onSubmit=**{**this**.*handleSubmit*} **className="login-form"**>

<**Item**>

{

*/\**

### ———————————————————————————

用户名*/*密码的的合法性要求

* + - * 1. 必须输入
        2. 必须大于等于*4* 位
        3. 必须小于等于*12* 位
        4. 必须是英文、数字或下划线组成

*\*/*

}

{

getFieldDecorator(**'username'**, { *//* 配置对象*:* 属性名是特定的一些名称

*//* 声明式验证*:* 直接使用别人定义好的验证规则进行验证

**rules**: [

{ **required**: **true**, **whitespace**: **true**, **message**: **'**用户名必须输入**'** },

{ **min**: 4, **message**: **'**用户名至少 **4** 位**'** },

{ **max**: 12, **message**: **'**用户名最多 **12** 位**'** },

{ **pattern**: /^[a-zA-Z0-9\_]+$/, **message**: **'**用户名必须是英文、数字或下划

线组成**'** },

],

**initialValue**: **'admin'** *//*指定初始值

})(

<**Input**

**prefix=**{<**Icon type="user" style=**{{ **color**: **'rgba(0,0,0,.25)'** }} />}

)

}

</**Item**>

<**Form.Item**>

{

getFieldDecorator(**'password'**, {

**rules**: [

{

**validator**: **this**.*validatePwd*

}

]

/>

**placeholder="**用户名**"**

})(

<Input

**ype="lock" style=**{{ **color**: **'rgba(0,0,0,.25)'** }} />}

|  |  |
| --- | --- |
| **prefix=**{ | <**Icon t** |
| **type="password"** | |

**placeholder="**密码**"**

/>

)

}

</**Form.Item**>

<Form.Item>

<Button type="primary" htmlType="submit" className="login-form-button">

登陆

</**Button**>

</**Form.Item**>

</**Form**>

</**section**>

</**div**>

)

}

}

**const** WrapLogin = Form.*create*()(Login)

**export default** WrapLogin

* + 1. **admin/admin.jsx**

**import** React, {Component} **from 'react' import** {Redirect} **from 'react-router-dom'**

**import** memeoryUtils **from '../../utils/memoryUtils'**

*/\**

后台管理的路由组件

*\*/*

**export default class** Admin **extends** Component { render () {

**const** user = memeoryUtils.**user if**(!user.**\_id**) {

return <Redirect to='/login'/>

}

return (

<**div**>

<**h2**>后台管理</**h2**>

<**div**>Hello {user.**username**}</**div**>

</**div**>

)

}

}

* 1. 维持登陆与自动登陆

*/\**

1. 登陆后*,* 刷新后依然是已登陆状态 *(*维持登陆*)*
2. 登陆后*,* 关闭浏览器后打开浏览器访问依然是已登陆状态 *(*自动登陆*)*
3. 登陆后*,* 访问登陆路径自动跳转到管理界面

*\*/*

* + 1. **utils/storageUtils.js**

**import** store **from 'store'**

**const** USER\_KEY = **'user\_key'**

*/\**

包含*n* 个操作*local storage* 的工具函数的模块

*\*/*

**export default** { saveUser(user) {

*// localStroage* 只能保存*string,* 如果传递是对象*,* 会自动调用对象的*toString()*并保存

*//localStorage.setItem(USER\_KEY, JSON.stringify(user)) //* 保存的必须是对象的*json* 串

store.set(USER\_KEY, user) *//* 内部会自动转换成*json* 再保存

},

getUser() { *//* 如果存在*,* 需要返回的是对象*,* 如果没有值*,* 返回*{}*

*// return JSON.parse(localStorage.getItem(USER\_KEY) || '{}') // [object, Object]*

**return** store.get(USER\_KEY) || {}

},

removeUser() {

*// localStorage.removeItem(USER\_KEY)*

store.remove(USER\_KEY)

}

}

* + 1. **login/login.jsx**

*//* 判断登录是否成功

**if** (result.**status** === 0) {

*//* 登录成功

*//* 提示登录成功，保存用户登录信息，跳转到主页面

message.success(**'**登录成功**'**);

*//* 保存用户数据

**const** user = result.**data** storageUtils.saveUser(user) memoryUtils.**user** = user

*//* 跳转到后台管理路由*(*已经登录成功，不需要回退了*)*

**this**.**props**.history.replace(**'/'**)

}

render() {

*//* 如果用户已经登陆*,* 自动跳转到*admin*

**if** (memoryUtils.user && memoryUtils.user.\_id) {

return <Redirect to='/'/>

}

}

* + 1. **src/index.js**

**import** storageUtils **from './utils/storageUtils' import** memoryUtils **from './utils/memoryUtils'**

*//* 如果*local* 中保存了*user,* 将*user* 保存到内存中

**const** user = storageUtils.getUser()

**if**(user && user.**\_id**) { memoryUtils.**user** = user

}

* 1. **Admin** 组件**(**搭建整体结构**)**
     1. 整体组件组成



* + 1. **LeftNav** 组件

#### components/left-nav/index.less

.left-nav {

**color**: **white**;

}

1. **components/left-nav/index.jsx**

**import** React, {Component} **from 'react'**

import './index.less'

*/\**

左侧导航组件

*\*/*

**export default class** LeftNav **extends** Component {

render() {

return (

<**div className="left-nav"**> LeftNav

</**div**>

)

}

}

* + 1. **Header** 组件

#### components/header/index.less

.header {

**height**: 80**px**;

}

1. **components/header/index.jsx**

**import** React, {Component} **from 'react'**

import './index.less'

*/\**

头部组件

*\*/*

**export default class** Header **extends** Component {

render() {

return (

<**div className='header'**> Header

</**div**>

)

}

}

* + 1. **Admin** 组件

#### 1) pages/admin/admin.jsx

**import** React, {Component} **from 'react' import** {Redirect} **from 'react-router-dom'**

**import** { Layout } **from 'antd'**

**import** memeoryUtils **from '../../utils/memoryUtils' import** Header **from '../../components/header' import** LeftNav **from '../../components/left-nav'**

**const** { Footer, Sider, Content } = Layout

*/\**

后台管理的路由组件

*\*/*

**export default class** Admin **extends** Component { render () {

**const** user = memeoryUtils.**user if**(!user.**\_id**) {

**return** <**Redirect to='/login'**/>

}

**return** (

<**Layout style=**{{**height**: **'100%'**}}>

<**Sider**>

<**LeftNav**/>

</**Sider**>

<**Layout**>

<**Header**>Header</**Header**>

<**Content style=**{{**backgroundColor**: **'white'**}}>Content</**Content**>

<**Footer style=**{{**textAlign**: **'center'**, **color**: **'#aaaaaa'**}}>推荐使用谷歌浏览器，可以获得更佳页面操作体验</**Footer**>

</**Layout**>

</**Layout**>

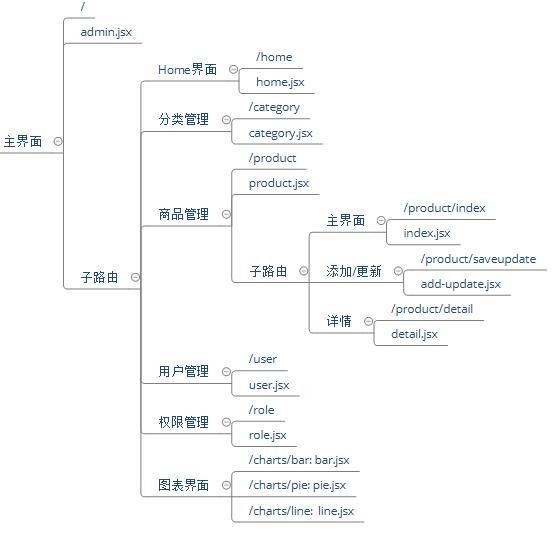
)

}

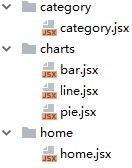
}

* 1. **Admin** 的子路由

### 组成



* + 1. 定义各个子路由组件



* + 1. 注册路由**: admin.jsx**

**import** React, {Component} **from 'react'**

**import** {Redirect, Route, Switch} **from 'react-router-dom' import** {Layout} **from 'antd'**

import memeoryUtils from '../../utils/memoryUtils' import Header from '../../components/header' import LeftNav from '../../components/left-nav' import Home from '../home/home'

import Category from '../category/category' import Product from '../product/product' import Role from '../role/role'

import User from '../user/user' import Bar from '../charts/bar' import Line from '../charts/line' import Pie from '../charts/pie'

**const** {Footer, Sider, Content} = Layout

*/\**

后台管理的路由组件

*\*/*

**export default class** Admin **extends** Component { render() {

**const** user = memeoryUtils.**user if** (!user.**\_id**) {

**return** <**Redirect to='/login'**/>

}

**return** (

<**Layout style=**{{**height**: **'100%'**}}>

<**Sider**>

<**LeftNav**/>

</**Sider**>

<**Layout**>

<**Header**>Header</**Header**>

<**Content style=**{{**backgroundColor**: **'white'**}}>

<**Switch**>

<**Route path='/home' component=**{Home}/>

<**Route path='/category' component=**{Category}/>

<**Route path='/product' component=**{Product}/>

<**Route path='/role' component=**{Role}/>

<**Route path='/user' component=**{User}/>

<**Route path='/charts/bar' component=**{Bar}/>

<**Route path='/charts/line' component=**{Line}/>

<**Route path='/charts/pie' component=**{Pie}/>

<**Redirect to='/home'** />

</**Switch**>

</**Content**>

<**Footer style=**{{**textAlign**: **'center'**, **color**: **'#aaaaaa'**}}>推荐使用谷歌浏览器，可以获得更佳页面操作体验</**Footer**>

</**Layout**>

</**Layout**>

)

}

}

* 1. **LeftNav** 组件
     1. 导航菜单配置**: config/menuConfig.js**

**const** menuList = [

{

**title**: **'**首页**'**, *//* 菜单标题名称 **key**: **'/home'**, *//* 对应的*path* **icon**: **'home'**, *//* 图标名称

},

{

**title**: **'**商品**'**, **key**: **'/products'**,

**icon**: **'appstore'**, **children**: [ *//* 子菜单列表

{

**title**: **'**品类管理**'**, **key**: **'/category'**,

**icon**: **'bars'**

},

{

**title**: **'**商品管理**'**, **key**: **'/product'**,

**icon**: **'tool'**

},

]

},

{

**title**: **'**用户管理**'**,

**key**: **'/user'**, **icon**: **'user'**

},

{

**title**: **'**角色管理**'**,

**key**: **'/role'**, **icon**: **'safety'**,

},

{

**title**: **'**图形图表**'**,

**key**: **'/charts'**, **icon**: **'area-chart'**, **children**: [

{

**title**: **'**柱形图**'**, **key**: **'/charts/bar'**,

icon: 'bar-chart'

},

{

**title**: **'**折线图**'**,

key: '/charts/line', icon: 'line-chart'

},

{

**title**: **'**饼图**'**,

**key**: **'/charts/pie'**, **icon**: **'pie-chart'**

},

]

},

]

**export default** menuList

## left-nav/index.less

.left-nav {

.logo-link {

**display**: **flex**;

**align-items**: **center**; **height**: 80**px**;

background-color: #002140; img {

**width**: 40**px**;

**height**: 40**px**;

**margin**: 0 15**px** 0 15**px**;

}

**h1** {

**margin-bottom**: 0;

### ———————————————————————————

**color**: **white**; **font-size**: 20**px**;

}

}

}

* + 1. 导航菜单组件**: left-nav/index.jsx**

**import** React, {Component} **from 'react'**

**import** {Link, withRouter} **from 'react-router-dom' import** {Menu, Icon} **from 'antd'**

**import** menuConfig **from '../../config/menuConfig' import** logo **from '../../assets/images/logo.png' import './index.less'**

**const** SubMenu = Menu.*SubMenu*

*/\**

左侧导航组件

*\*/*

**class** LeftNav **extends** Component {

*/\**

根据指定菜单数据列表产生*<Menu>*的子节点数组使用*reduce() +* 递归

*\*/*

*getMenuNodes* = (menuList) => {

*//* 得到当前请求的*path*

**const** path = **this**.**props**.location.**pathname**

**return** menuList.reduce((pre, item) => {

**if** (!item.**children**) { pre.push((

<**Menu.Item key=**{item.**key**}>

<**Link to=**{item.**key**}>

<**Icon type=**{item.**icon**}/>

<**span**>{item.**title**}</**span**>

</**Link**>

</**Menu.Item**>

### ———————————————————————————

))

} **else** { pre.push((

<**SubMenu key=**{item.**key**}

**title=**{

<**span**>

<**Icon type=**{item.**icon**}/>

<**span**>{item.**title**}</**span**>

</**span**>

}

>

{**this**.*getMenuNodes*(item.**children**)}

</**SubMenu**>

))

*//* 如果当前请求路由与当前菜单的某个子菜单的*key* 匹配*,* 将菜单的*key* 保存为*openKey*

**if**(item.**children**.find(cItem => path.indexOf(cItem.**key**)===0)) {

**this**.**openKey** = item.**key**

}

}

**return** pre

}, [])

}

*/\**

根据指定菜单数据列表产生*<Menu>*的子节点数组使用 *map() +* 递归

*\*/*

*getMenuNodes2* = (menuList) => {

*//* 得到当前请求的*path*

**const** path = **this**.**props**.location.**pathname**

**return** menuList.map(item => {

**if**(!item.**children**) {

return (

<**Menu.Item key=**{item.**key**}>

<**Link to=**{item.**key**}>

<**Icon type=**{item.**icon**}/>

<**span**>{item.**title**}</**span**>

</**Link**>

### ———————————————————————————

</**Menu.Item**>

)

} **else** {

*//* 如果当前请求路由与当前菜单的某个子菜单的*key* 匹配*,* 将菜单的*key* 保存为*openKey*

**if**(item.**children**.find(cItem => path.indexOf(cItem.**key**)===0)) {

**this**.**openKey** = item.**key**

}

return (

<SubMenu

{item.**key**}

**title=**{

<**span**>

<**Icon type=**{item.**icon**}/>

<**span**>{item.**title**}</**span**>

</**span**>

}

>

{**this**.*getMenuNodes*(item.**children**)}

</**SubMenu**>

)

}

})

}

*/\**

在第一次*render()*之前执行一次

一般可以在此同步为第一次*render()*准备数据

*\*/*

componentWillMount() {

*// this.menuNodes = this.getMenuNodes(menuConfig)*

**this**.**menuNodes** = **this**.*getMenuNodes2*(menuConfig)

}

render() {

**key=**

*//* 得到当前请求路径*,* 作为选中菜单项的 *key*

**const** selectKey = **this**.**props**.location.**pathname const** openKey = **this**.**openKey**

return (

<div className="left-nav">

<Link to='/home' className='logo-link'>

<**img src=**{logo} **alt="logo"**/>

<**h1**>硅谷后台</**h1**>

</**Link**>

<**Menu**

**mode="inline"**

**theme="dark" selectedKeys=**{[selectKey]}

**defaultOpenKeys=**{[openKey]}

{

**this**.**menuNodes**

}

</**Menu**>

</**div**>

)

}

}

*/\**

*withRouter:* 高阶组件*:* 包装非路由组件返回一个包装后的新组件*,* 新组件会向被包装组件传递

*history/location/match* 属性

*\*/*

**export default** withRouter(LeftNav)

*/\**

*2* 个问题*:*

1. 自动选中对应的菜单项
2. 有可能需要自动菜单项

*\*/*

>

* 1. **Header** 组件

### 下载依赖

**yarn add jsonp**

* + 1. **api/index.js**

###### 百度地图天气预报在线接口:

[**http://api.map.baidu.com/telematics/v3/weather?location=xxx&ou**](http://api.map.baidu.com/telematics/v3/weather?location=xxx&ou)

**tput=json&ak=3p49MVra6urFRGOT9s8UBWr2**

**import** jsonp **from 'jsonp'**

*/\**

通过*jsonp* 请求获取天气信息

*\*/*

**export function** *reqWeather*(city) {

**const** url =

`[http://api.map.baidu.com/telematics/v3/weather?location=$](http://api.map.baidu.com/telematics/v3/weather?location=%24){city}&output=json&ak=3p4 9MVra6urFRGOT9s8UBWr2`

**return new** Promise((resolve, reject) => { jsonp(url, {

param: 'callback'

}, (error, response) => {

**if** (!error && response.**status** == **'success'**) {

**const** {dayPictureUrl, weather} = response.**results**[0].weather\_data[0] resolve({dayPictureUrl, weather})

} **else** {

alert(**'**获取天气信息失败**'**)

}

})

})

}

* + 1. **header/index.less**

.header {

**height**: 80**px**;

background-color: #fff;

.header-top {

**height**: 40**px**;

**line-height**: 40**px**; **text-align**: **right**; **padding-right**: 20**px**;

border-bottom: 1px solid #1DA57A;

}

.**header-bottom** {

**display**: **flex**;

**align-items**: **center**; **height**: 40**px**;

.**header-bottom-left** { **position**: **relative**; **width**: 25%;

**font-size**: 20**px**; **text-align**: **center**; **&**::**after** {

**content**: **''**; **position**: **absolute**; **top**: 30**px**;

**right**: 50%;

**transform**: **translateX**(50%); **border-top**: 20**px solid white**;

**border-right**: 20**px solid transparent**; **border-bottom**: 20**px solid transparent**; **border-left**: 20**px solid transparent**;

}

}

.**header-bottom-right** {

**width**: 75%;

**text-align**: **right**; **margin-right**: 30**px**; **img** {

**width**: 30**px**; **height**: 20**px**; **margin**: 0 15**px**;

}

}

}

}

* + 1. **utils/dateUtils.js**

*/\**

包含*n* 个日期时间处理的工具函数模块

*\*/*

*/\**

格式化日期

*\*/*

**export function** *formateDate*(time) {

**if** (!time) **return ''**

**let** date = **new** Date(time)

**return** date.getFullYear() + **'-'** + (date.getMonth() + 1) + **'-'** + date.getDate()

+ **' '** + date.getHours() + **':'** + date.getMinutes() + **':'** + date.getSeconds()

}

* + 1. 抽取通用组件**: link-button 1). index.jsx**

import React from 'react' import './index.less'

*/\**

通用的看起来像链接的*button* 组件

*\*/*

**export default function** LinkButton (props) {

**return** <**button** {...props} **className='link-button'**></**button**>

}

**2). index.less**

.**link-button** { **border**: **none**; **outline**: **none**;

background-color: transparent; color: #1DA57A;

**cursor**: **pointer**;

}

* + 1. **header/header.jsx**

**import** React, {Component} **from 'react' import** {Modal} **from 'antd'**

**import** {withRouter} **from 'react-router-dom'**

**import** *LinkButton* **from '../link-button'**

**import** menuList **from '../../config/menuConfig' import** {*reqWeather*} **from '../../api'**

**import** {*formateDate*} **from '../../utils/dateUtils' import** memoryUtils **from '../../utils/memoryUtils' import** storageUtils **from '../../utils/storageUtils' import './index.less'**

*/\**

头部组件

*\*/*

**class** Header **extends** Component {

**state** = {

**sysTime**: *formateDate*(Date.now()), **dayPictureUrl**: **''**, *//* 天气图片的*url* **weather**: **''**

}

*/\**

发异步*ajax* 获取天气数据并更新状态

*\*/*

*getWeather* = **async** () => {

**const** {dayPictureUrl, weather} = **await** *reqWeather*(**'**北京**'**)

**this**.setState({ dayPictureUrl, weather

})

}

*/\**

启动循环定时器*,* 每隔*1s* 更新一次*sysTime*

*\*/*

*getSysTime* = () => {

**this**.**intervalId** = *setInterval*(() => {

**this**.setState({

**sysTime**: *formateDate*(Date.now())

})

}, 1000)

}

*/\**

退出登陆

*\*/*

*logout* = () => {

Modal.*confirm*({

**content**: **'**确定退出吗**?'**, onOk: () => {

***console***.log(**'OK'**)

*//* 移除保存的 *user*

storageUtils.removeUser() memoryUtils.**user** = {}

*//* 跳转到*login*

**this**.**props**.history.replace(**'/login'**)

},

onCancel() {

***console***.log(**'Cancel'**)

},

})

}

*/\**

根据请求的*path* 得到对应的标题

*\*/*

*getTitle* = (path) => {

**let** title menuList.forEach(menu => {

**if**(menu.**key**===path) {

title = menu.**title**

} **else if** (menu.**children**) { menu.**children**.forEach(item => {

**if**(path.indexOf(item.**key**)===0) { title = item.**title**

}

})

}

})

### ———————————————————————————

**return** title

}

componentDidMount () { **this**.*getSysTime*() **this**.*getWeather*()

}

componentWillUnmount () {

*//* 清除定时器

*clearInterval*(**this**.**intervalId**)

}

render() {

**const** {**sysTime**, **dayPictureUrl**, **weather**} = **this**.**state**

*//* 得到当前用户

**const** user = memoryUtils.**user**

*//* 得到当前请求的路径

**const** path = **this**.**props**.**location**.**pathname**

*//* 得到对应的标题

**const** title = **this**.*getTitle*(path)

**return** (

<**div className="header"**>

<**div className="header-top"**>

<**span**>欢迎, {user.**username**}</**span**>

<**LinkButton onClick=**{**this**.*logout*}>退出</**LinkButton**>

</**div**>

<**div className="header-bottom"**>

<**div className="header-bottom-left"**>{title}</**div**>

<**div className="header-bottom-right"**>

<**span**>{**sysTime**}</**span**>

<img src={dayPictureUrl} alt="weather"/>

<**span**>{**weather**}</**span**>

</**div**>

</**div**>

</**div**>

)

}

**export default** withRouter(Header)

## pages/admin/admin.jsx

<**Content style=**{{**backgroundColor**: **'white'**, **margin**: **'20px 20px 0'**}}>

* 1. **Home** 界面

## home.less

.**home**{

**width**: 100%;

**height**: 100%;

**display**: **flex**;

**align-items**: **center**; **justify-content**: **center**; **font-size**: 30**px**;

}

**2.15.1. home.jsx**

import React from 'react' import './home.less'

*/\**

*Home* 路由组件

*\*/*

**export default function** Home(props) {

return (

<div className="home">

欢迎使用硅谷后台管理系统

</**div**>

)

* 1. 分类管理
     1. **api/index.js**

*//* 获取一级或某个二级分类列表

**export const** *reqCategorys* = (parentId) => *ajax*(**'/manage/category/list'**, {parentId})

*//* 添加分类

**export const** *reqAddCategory* = (parentId, categoryName) => *ajax*(**'/manage/category/add'**,

{

parentId, categoryName

}, **'POST'**)

*//* 更新品类名称

**export const** *reqUpdateCategory* = ({categoryId, categoryName}) =>

*ajax*(**'/manage/category/update'**, { categoryId,

categoryName

}, **'POST'**)

* + 1. **category.jsx**

**import** React, {Component} **from 'react' import** {

Card, Table, Button, Icon, message, Modal

} **from 'antd'**

**import** UpdateForm **from './update-form'**

### ———————————————————————————

**import** AddForm **from './add-form'**

**import** *LinkButton* **from '../../components/link-button'**

**import** {*reqCategorys*, *reqAddCategory*, *reqUpdateCategory*} **from "../../api"**;

*/\**

分类管理路由组件

*\*/*

**export default class** Category **extends** Component {

**state** = {

**categorys**: [], *//* 一级分类列表 **subCategorys**: [], *//* 二级分类列表 **parentId**: **'0'**, *//* 父分类的*ID* **parentName**: **''**, *//* 父分类的名称 **loading**: **false**, *//* 标识是否正在加载中

**showStatus**: 0, *//* 是否显示对话框 *0:* 都不显示*, 1:* 显示添加*, 2:* 显示更新

}

*/\**

根据*parentId* 异步获取分类列表显示

*\*/*

*getCategorys* = **async** (parentId) => {

*//* 更新*loading* 状态*:* 加载中

**this**.setState({ **loading**: **true**

})

*//* 优先使用指定的*parentId,* 如果没有指定使用状态中的*parentId*

parentId = parentId || **this**.**state**.**parentId**

*//* 异步获取分类列表

**const** result = **await** *reqCategorys*(parentId) *// {status: 0, data: []}*

*//* 更新*loading* 状态*:* 加载完成

**this**.setState({

**loading**: **false**

})

**if** (result.**status** === 0) { **const** categorys = result.**data if** (parentId === **'0'**) {

*//* 更新一级分类列表

**this**.setState({ categorys

})

### ———————————————————————————

} **else** {

*//* 更新二级分类列表

**this**.setState({ **subCategorys**: categorys

})

}

} **else** {

*//* 获取列表失败

message.error(**'**获取列表失败**'**)

}

}

*/\**

显示指定分类的子分类列表

*\*/*

*showSubCates* = (category) => {

*// console.log('set* 之前*', this.state.parentId) // 0*

*//* 更新状态*: state* 中的数据是异步更新*(*不会立即更新*state* 中的数据*)*

**this**.setState({

**parentId**: category.**\_id**,

**parentName**: category.**name**

}, () => { *//* 在状态更新之后执行*,* 在回调函数中能得到最新的状态数据

**this**.*getCategorys*()

})

*// console.log('set* 之后*', this.state.parentId) // xxx*

}

*/\**

显示一级列表

*\*/*

*showCategorys* = () => {

**this**.setState({ **parentId**: **'0'**,

**parentName**: **''**, **subCategorys**: [], **showStatus**: 0,

})

}

*/\**

显示添加的对话框

### ———————————————————————————

*\*/*

*showAdd* = () => { **this**.setState({ **showStatus**: 1

})

}

*/\**

显示修改的对话框

*\*/*

*showUpdate* = (category) => {

*//* 保存*category*

**this**.**category** = category

*//* 更新状态

**this**.setState({ **showStatus**: 2

})

}

*/\**

添加分类

*\*/*

*addCategory* = **async** () => {

*//* 得到数据

**const** {parentId, categoryName} = **this**.**form**.getFieldsValue()

*//* 关闭对话框

**this**.setState({ **showStatus**: 0

})

*//* 重置表单

**this**.**form**.resetFields()

*//* 异步请求添加分类

**const** result = **await** *reqAddCategory*(categoryName, parentId)

**if** (result.**status** === 0) {

*/\**

添加一级分类

在当前分类列表下添加

*\*/*

**if**( parentId===**this**.**state**.**parentId**) {

**this**.*getCategorys*()

} **else if** (parentId === **'0'**) {

### ———————————————————————————

**this**.*getCategorys*(parentId)

}

}

}

*/\**

更新分类

*\*/*

*updateCategory* = **async** () => {

*//* 得到数据

**const** categoryId = **this**.**category**.**\_id**

**const** {categoryName} = **this**.**form**.getFieldsValue()

*//* 关闭对话框

**this**.setState({ **showStatus**: 0

})

*//* 重置表单

**this**.**form**.resetFields()

*//* 异步请求更新分类

**const** result = **await** *reqUpdateCategory*({categoryId, categoryName})

**if** (result.**status** === 0) {

*//* 重新获取列表

**this**.*getCategorys*()

}

}

componentWillMount() {

**this**.**columns** = [

{

**title**: **'**分类名称**'**, **dataIndex**: **'name'**,

},

{

**title**: **'**操作**'**, **width**: 300,

render: (category) => (

<**span**>

<**LinkButton onClick=**{() => **this**.*showUpdate*(category)}>修改分类

</**LinkButton**>**&nbsp;&nbsp;&nbsp;**

### ———————————————————————————

{**this**.**state**.**parentId** === **'0'** ?

<**LinkButton onClick=**{() => **this**.*showSubCates*(category)}>查看子分类</**LinkButton**> : **null**}

</**span**>

)

}];

}

componentDidMount() {

**this**.*getCategorys*()

}

render() {

*//* 从状态中取数据

**const** {**categorys**, **subCategorys**, **parentId**, **parentName**, **loading**, **showStatus**} = **this**.**state**

*//* 从组件对象中数据

**const** category = **this**.**category** || {}

*// Card* 的左侧标题

**const** title = **parentId** === **'0'** ? **'**一级分类列表**'** : (

<**span**>

<**LinkButton onClick=**{**this**.*showCategorys*}>一级分类列表

<**Icon type='arrow-right'**/>**&nbsp;&nbsp;**

<**span**>{**parentName**}</**span**>

</**span**>

)

*// Card* 的右侧*button*

**const** extra = (

<**Button type='primary' onClick=**{**this**.*showAdd*}>

添加

</**Button**>

)

**return** (

<**Card title=**{title} **extra=**{extra}>

**&nbsp;&nbsp;**

</**LinkButton**>

<**Icon type='plus'**/>

<**Table**

**bordered**

**rowKey='\_id'**

**dataSource=**{**parentId** === **'0'** ? **categorys** : **subCategorys**}

**columns=**{**this**.**columns**}

**loading=**{**loading**}

**pagination=**{{**pageSize**: 5, **showQuickJumper**: **true**, **showSizeChanger**: **true**}}

/>

### ———————————————————————————

**title="**添加分类**"**

**visible=**{**showStatus** === 1}

**onOk=**{**this**.*addCategory*}

**onCancel=**{() => **this**.setState({**showStatus**: 0})}

>

<**AddForm categorys=**{**categorys**}

**parentId=**{**parentId**}

**setForm=**{form => **this**.**form** = form}

/>

</**Modal**>

<**Modal**

**title="**修改分类**"**

**visible=**{**showStatus** === 2}

**onOk=**{**this**.*updateCategory*}

**onCancel=**{() => {

**this**.setState({**showStatus**: 0})

**this**.**form**.resetFields()

}}

>

<**UpdateForm**

**categoryName=**{category.**name**}

**setForm=**{form => **this**.**form** = form}

/>

</**Modal**>

</**Card**>

)

}

}

<**Modal**

* + 1. **add-form.jsx**

**import** React, {Component} **from 'react'**

**import** {Form, Select, Input} **from 'antd' import** PropTypes **from 'prop-types'**

———————————————————————————

**const** Item = Form.*Item*

**const** Option = Select.***Option***

*/\**

添加分类的*Form* 组件

*\*/*

**class** AddForm **extends** Component {

**static** *propTypes* = {

**categorys**: PropTypes.array.isRequired, **parentId**: PropTypes.string.isRequired, **setForm**: PropTypes.func.isRequired,

}

componentWillMount() {

**this**.**props**.**setForm**(**this**.**props**.**form**)

}

render() {

**const** {getFieldDecorator} = **this**.**props**.**form**

**const** {**categorys**, **parentId**} = **this**.**props return** (

<**Form**>

<**Item label='**所属分类**'**>

{

getFieldDecorator(**'parentId'**, {

**initialValue**: **parentId**

})(

<**Select**>

<**Option key='0' value='0'**>一级分类</**Option**>

{

**categorys**.map(c => <**Option key=**{c.**\_id**}

**value=**{c.**\_id**}>{c.**name**}</**Option**>)

}

</**Select**>

)

}

</**Item**>

<**Item label='**分类名称**'**>

### ———————————————————————————

{

getFieldDecorator(**'categoryName'**, {

initialValue: ''

})(

<**Input placeholder='**请输入分类名称**'**/>

)

}

</**Item**>

</**Form**>

)

}

}

**export default** AddForm = Form.*create*()(AddForm)

* + 1. **update-form.jsx**

**import** React, {Component} **from 'react' import** {Form, Input} **from 'antd' import** PropTypes **from 'prop-types'**

**const** Item = Form.*Item*

*/\**

更新分类的*Form* 组件

*\*/*

**class** UpdateForm **extends** Component {

**static** *propTypes* = {

**categoryName**: PropTypes.string,

**setForm**: PropTypes.func.isRequired,

}

componentWillMount() {

this.props.setForm(this.props.form)

}

render() {

———————————————————————————

**const** {getFieldDecorator} = **this**.**props**.**form const** {**categoryName**} = **this**.**props**

return (

<**Form**>

<**Item**>

{

getFieldDecorator(**'categoryName'**, {

initialValue: categoryName

})(

<**Input placeholder='**请输入分类名称**'**/>

)

}

</**Item**>

</**Form**>

)

}

}

**export default** UpdateForm = Form.*create*()(UpdateForm)

* 1. 商品管理
     1. 下载依赖

**yarn add react-draft-wysiwyg draftjs-to-html**

* + 1. **api/index.js**

*//* 根据分类*ID* 获取分类

**export const** *reqCategory* = (categoryId) => *ajax*(**'/manage/category/info'**, {categoryId})

*//* 获取商品分页列表

**export const** *reqProducts* = (pageNum, pageSize) => *ajax*(**'/manage/product/list'**, {pageNum, pageSize})

*//* 根据*ID/Name* 搜索产品分页列表

———————————————————————————

**export const** *reqSearchProducts* = ({pageNum, pageSize, searchType, searchName}) =>

*ajax*(**'/manage/product/search'**, { pageNum,

pageSize,

[searchType]: searchName,

})

*//* 添加*/*更新商品

**export const** *reqAddOrUpdateProduct* = (product) => *ajax*(**'/manage/product/'** + (product.**\_id** ? **'update'** : **'add'**), product, **'post'**)

*//* 对商品进行上架*/*下架处理

**export const** *reqUpdateProductStatus* = (productId, status) =>

*ajax*(**'/manage/product/updateStatus'**, { productId,

status

}, **'POST'**)

*//* 删除图片

**export const** *reqDeleteImg* = (name) => *ajax*(**'/manage/img/delete'**, {name}, **'post'**)

* + 1. **product/product.jsx**

**import** React, {Component} **from 'react'**

**import** {Switch, Route, Redirect} **from 'react-router-dom'**

**import** ProductHome **from './home'**

**import** ProductAddUpdate **from './add-update' import** ProductDetail **from './detail'**

import './product.less'

*/\**

管理的商品管理路由组件

*\*/*

**export default class** Product **extends** Component { render() {

return (

<**Switch**>

<**Route path='/product' exact component=**{ProductHome}/>

———————————————————————————

<**Route path='/product/addupdate' component=**{ProductAddUpdate}/>

<**Route path='/product/detail' component=**{ProductDetail}/>

<Redirect to='/product'/>

</**Switch**>

)

}

}

* + 1. **product/home.jsx**

**import** React, {Component} **from 'react' import** {

Card, Select, Input, Button, Icon, Table, message

} **from 'antd'**

**import** *LinkButton* **from '../../components/link-button'**

**import** {*reqProducts*, *reqSearchProducts*, *reqUpdateProductStatus*} **from '../../api' import** {PAGE\_SIZE} **from '../../utils/constants'**

**const** Option = Select.***Option***

*/\**

商品管理的主界面路由

*\*/*

**export default class** ProductHome **extends** Component {

**state** = {

**total**: 0, *//* 商品的总数量

**products**: [], *//* 当前页列表数据

**searchType**: **'productName'**, *//* 搜索类型 *productName / productDesc*

**searchName**: **''**, *//* 搜索关键字

}

———————————————————————————

*/\**

更新指定产品的状态

*\*/*

*updateProductStatus* = **async** (productId, status) => {

**const** result = **await** *reqUpdateProductStatus*(productId, status)

**if** (result.**status** === 0) { message.success(**'**更新状态成功**!'**) **this**.*getProducts*(**this**.**pageNum** || 1)

}

}

*/\**

初始化生成*Tabe* 所有列的数组

*\*/*

*initColumns* = () => {

**this**.**columns** = [

{

**title**: **'**商品名称**'**, **dataIndex**: **'name'**

},

{

**title**: **'**商品描述**'**, **dataIndex**: **'desc'**

},

{

**title**: **'**价格**'**, **dataIndex**: **'price'**,

render: (price) => <**span**>¥{price}</**span**>

},

{

**title**: **'**状态**'**, **width**: 100,

**dataIndex**: **'status'**,

render: (status, product) => { *// 1:* 在售*, 2:* 已下架

**let** btnText = **'**下架**'**

**let** statusText = **'**在售**'**

**if** (status === 2) { btnText = **'**上架**'**

statusText = **'**已下架**'**

}

### ———————————————————————————

status = status === 1 ? 2 : 1

return (

<**span**>

<Button type='primary' onClick={() =>

**this**.*updateProductStatus*(product.**\_id**, status)}>{btnText}</**Button**>

<**span**>{statusText}</**span**>

</**span**>

)

}

},

{

**title**: **'**操作**'**, **width**: 100,

render: (product) => (

<**span**>

<**LinkButton onClick=**{() => **this**.**props**.history.push(**'/product/detail'**, product)}>详情</**LinkButton**>

**&nbsp;&nbsp;&nbsp;**

<**LinkButton onClick=**{() => **this**.**props**.history.push(**'/product/addupdate'**, product)}>修改</**LinkButton**>

</**span**>

)

},

]

}

*/\**

异步获取指定页的数据

*\*/*

*getProducts* = **async** (pageNum) => {

**this**.**pageNum** = pageNum

**const** {**searchType**, **searchName**} = **this**.**state**

**let** result

**if** (**searchName**) { *//* 搜索分页

result = **await** *reqSearchProducts*({pageNum, **pageSize**: PAGE\_SIZE, **searchType**, **searchName**})

} **else** { *//* 一般分页

result = **await** *reqProducts*(pageNum, PAGE\_SIZE)

}

***console***.log(**'getProducts()'**, result)

### ———————————————————————————

**if** (result.**status** === 0) {

**const** {total, list} = result.**data this**.setState({

total,

**products**: list

})

}

}

componentWillMount() {

**this**.*initColumns*()

}

componentDidMount() {

**this**.*getProducts*(1)

}

render() {

**const** {**products**, **total**, **searchType**} = **this**.**state**

**const** title = (

<**span**>

<**Select value=**{**searchType**} **onChange=**{value => **this**.setState({**searchType**: value})}>

<**Option key='productName' value='productName'**>按名称搜索</**Option**>

<**Option key='productDesc' value='productDesc'**>按描述搜索</**Option**>

</**Select**>

<**Input style=**{{**width**: 150, **marginLeft**: 10, **marginRight**: 10}} **placeholder='**

关键字**'**

**onChange=**{(e) => **this**.setState({**searchName**: e.**target**.**value**})}/>

<**Button type='primary' onClick=**{() => **this**.*getProducts*(1)}>搜索</**Button**>

</**span**>

)

**const** extra = (

<**Button type='primary' style=**{{**float**: **'right'**}} **onClick=**{() =>

**this**.**props**.history.push(**'/product/addupdate'**)}>

<Icon type='plus'/>

添加商品

</**Button**>

)

### ———————————————————————————

**return** (

<**div**>

<**Card title=**{title} **extra=**{extra}>

<**Table**

**rowKey='\_id'**

**columns=**{**this**.**columns**} **dataSource=**{**products**} **pagination=**{{

**defaultPageSize**: PAGE\_SIZE,

**total**, **showQuickJumper**: **true**,

**onChange**: **this**.*getProducts*

}}

/>

</**Card**>

</**div**>

)

}

}

**bordered**

* + 1. **product/detail.jsx**

**import** React, {Component} **from 'react' import** {List, Icon, Card} **from 'antd'**

**import** {*reqCategory*} **from '../../api'**

**import** {BASE\_IMG\_PATH} **from '../../utils/constants' import** *LinkButton* **from "../../components/link-button"**;

*/\**

商品详情组件

*\*/*

**export default class** ProductDetail **extends** Component {

**state** = {

**cName1**: **''**, *//* 一级分类名称

———————————————————————————

**cName2**: **''** *//* 二级分类名称

}

*/\**

异步获取当前产品对应的分类名称

*\*/*

*getCategoryName* = **async** () => {

**const** {categoryId, pCategoryId} = **this**.**props**.location.**state if** (pCategoryId === **'0'**) {

*//* 获取一级分类名称

**const** result = **await** *reqCategory*(categoryId) **const** cName1 = result.**data**.**name this**.setState({cName1})

} **else** {

*//* 获取一级分类名称

*/\*const result1 = await reqCategory(pCategoryId) const cName1 = result1.data.name*

*//* 获取二级分类名称

*const result2 = await reqCategory(categoryId) const cName2 = result2.data.name this.setState({cName1, cName2})\*/*

*/\**

一次发多个请求*,* 等所有请求都返回后一起处理*,* 如果有一个请求出错了*,* 整个都会失败

*Promise.all([promise1, promise2])* 返回值一个*promise* 对象*,* 异步成功返回的是

*[result1, result2]*

*\*/*

**const** results = **await** Promise.all([*reqCategory*(pCategoryId),

*reqCategory*(categoryId)])

**const** result1 = results[0]

**const** result2 = results[1]

**const** cName1 = result1.**data**.**name const** cName2 = result2.**data**.**name this**.setState({cName1, cName2})

}

}

componentDidMount() {

**this**.*getCategoryName*()

}

render() {

### ———————————————————————————

**const** {name, desc, price, imgs, detail} = **this**.**props**.location.**state**

**const** {**cName1**, **cName2**} = **this**.**state const** title = (

<**span**>

<**LinkButton onClick=**{() => **this**.**props**.history.*goBack*()}>

<**Icon type="arrow-left" style=**{{**fontSize**: 20}}/>

</**LinkButton**>

**&nbsp;&nbsp;**商品详情

</**span**>

)

**const** imgStyle = {**width**: 150, **height**: 150, **marginRight**: 10, **border**: **'1px solid black'**}

**return** (

<**Card className='product-detail' title=**{title}>

<**List**>

商品名称:</**span**>

<**span**>{name}</**span**>

</**List.Item**>

商品描述:</**span**>

<**span**>{desc}</**span**>

</**List.Item**>

商品价格:</**span**>

<**span**>{price + **'**元**'**}</**span**>

</**List.Item**>

所属分类:</**span**>

<**span**>{**cName1** + (**cName2** ? **' --> '** + **cName2** : **''**)}</**span**>

</**List.Item**>

商品图片:</**span**>

<**span**>

<**span className='left'**>

<**List.Item**>

<**span className='left'**>

<**List.Item**>

<**span className='left'**>

<**List.Item**>

<**span className='left'**>

<**List.Item**>

<**span className='left'**>

<**List.Item**>

{

imgs.map(img => (

<**img src=**{BASE\_IMG\_PATH + img} **alt="img" key=**{img} **style=**{imgStyle}/>

))

}

</**span**>

</**List.Item**>

<List.Item>

<**span className='left'**>商品详情:</**span**>

<**div dangerouslySetInnerHTML=**{{ **html**: detail}}></**div**>

</**List.Item**>

</**List**>

</**Card**>

)

}

}

## product/add-update.jsx

**import** React, {Component} **from 'react' import** {

Card, Icon, Form, Input, Cascader, Button, message

} **from 'antd'**

**import** *LinkButton* **from '../../components/link-button'**

**import** PicturesWall **from './pictures-wall' import** RichTextEditor **from './rich-text-editor'**

**import** {*reqCategorys*, *reqAddOrUpdateProduct*} **from '../../api'**

**const** {Item} = Form

**const** {TextArea} = Input

*/\**

商品添加*/*更新的路由组件

*\*/*

**class** ProductAddUpdate **extends** Component {

**state** = {

**options**: [], *//* 用来显示级联列表的数组

}

constructor(props) {

**super**(props);

**this**.**pw** = React.createRef();

**this**.**editor** = React.createRef();

}

*/\**

选择某个分类项时的回调加载对应的二级分类显示

*\*/*

*loadData* = **async** (selectedOptions) => {

*// console.log('loadDate()', selectedOptions)*

**const** targetOption = selectedOptions[selectedOptions.**length** - 1] targetOption.**loading** = **true** *//* 显示*loading*

———————————————————————————

*//* 异步请求获取对应的二级分类列表

**const** subCategorys = **await this**.*getCategorys*(targetOption.**value**) *// await* 的作用*:* 保证完成执行完保存的分类数组才进入后面的语句

targetOption.**loading** = **false** *//* 隐藏*loading*

**if**(subCategorys && subCategorys.**length**>0) { *//* 有子分类

*//* 生成一个二级的*options*

**const** cOptions = subCategorys.map(c => ({

**value**: c.**\_id**,

**label**: c.**name**,

**isLeaf**: **true**,

}))

*//* 添加为对应的*option* 的*children(*子*options)*

targetOption.**children** = cOptions

} **else** { *//* 没有子分类

targetOption.**isLeaf** = **true**

}

*//* 更新*options* 状态

**this**.setState({

**options**: [...**this**.**state**.**options**],

});

}

*/\**

获取指定分类*id* 的子分类列表

如果*parentId* 为*0* 时获取一级列表

*\*/*

*getCategorys* = **async** (parentId) => {

**const** result = **await** *reqCategorys*(parentId)

**if**(result.**status**===0) {

**const** categorys = result.**data**

**if**(parentId===**'0'**) {

*//* 根据一级分类数组初始化生成*options* 数组

**this**.*initOptions*(categorys)

} **else** { *//* 当前得到是二级分类列表

*//* 返回二级分类列表*(*作为*async* 函数的*promise* 对象的成功的*value* 值*)*

**return** categorys

}

}

### ———————————————————————————

}

*/\**

生成级联的一级列表

*\*/*

*initOptions* = **async** (categorys) => {

*//* 根据一级分类数组生成*option* 的数组

**const** options = categorys.map(c => ({

**value**: c.**\_id**,

**label**: c.**name**,

**isLeaf**: **false**,

}))

*//* 如果当前是更新*,* 且商品是一个二级分类的商品

**const** {**product**, **isUpdate**} = **this**

**if**(**isUpdate** && **product**.**pCategoryId**!==**'0'**) {

*//* 异步获取*product.pCategoryId* 的二级分类列表

**const** subCategorys = **await this**.*getCategorys*(**product**.**pCategoryId**) **if** (subCategorys && subCategorys.**length**>0) {

*//* 生成二级的*option* 数组

**const** cOptions = subCategorys.map(c => ({

**value**: c.**\_id**, **label**: c.**name**, **isLeaf**: **true**,

}))

*//* 找到对应的*option*

**const** targetOption = options.find(option => option.**value**===**product**.**pCategoryId**)

*//* 将*cOptions* 添加为对应的一级*option* 的*children*

targetOption.**children** = cOptions

}

}

*//* 更新状态

**this**.setState({ options

})

}

*/\**

### ———————————————————————————

对商品价格进行自定义验证

*\*/*

*validatePrice* = (rule, value, callback) => { value = value \* 1

**if**(value>0) { callback()

} **else** {

callback(**'**价格必须是大于 **0** 的数值**'**)

}

}

*//* 添加*/*更新

*submit* = () => {

**this**.**props**.form.validateFields(**async** (err, values) => {

**if** (!err) {

*//* 收集产品相关信息

**const** {name, desc, price, categoryIds} = values

*//* 在父组件中得到子组件对象*,* 调用子组件对象的方法

**const** imgs = **this**.**pw**.**current**.*getImgs*()

**const** detail = **this**.**editor**.**current**.*getDetail*()

**let** pCategoryId = **''**

**let** categoryId = **''**

**if**(categoryIds.**length**===1) { *//* 选择的是一级分类

pCategoryId = **'0'**

categoryId = categoryIds[0]

} **else** { *//* 选择的是二级分类

pCategoryId = categoryIds[0] categoryId = categoryIds[1]

}

*//* 封装成对象

**const** product = {name, desc, price, pCategoryId, categoryId, detail, imgs}

*//* 如果是更新*,* 指定*product* 的*\_id* 属性值

**if**(**this**.**isUpdate**) {

product.**\_id** = **this**.**product**.**\_id**

}

*//* 请求保存

**const** result = **await** *reqAddOrUpdateProduct*(product)

**if**(result.**status**===0) { message.success(**'**保存商品成功**'**) **this**.**props**.history.*goBack*()

} **else** {

### ———————————————————————————

message.success(**'**保存商品失败**'**)

}

}

})

}

componentDidMount () {

*//* 异步获取一级分类列表

**this**.*getCategorys*(**'0'**)

}

componentWillMount () {

*//* 取出跳转传入的数据

**const** product = **this**.**props**.location.**state this**.**product** = product || {}

**this**.**isUpdate** = !!product *// !!xxx* 将一个数据强制转换成布尔类型

}

render() {

**const** {**product**, **isUpdate**} = **this**

**const** {**pCategoryId**, **categoryId**} = **product const** {**options**} = **this**.**state**

**const** {getFieldDecorator} = **this**.**props**.form

*//* 准备用于级联列表显示的数组

**const** categoryIds = []

**if**(**isUpdate**) {

**if**(**pCategoryId**===**'0'**) { categoryIds.push(**categoryId**)

} **else** { categoryIds.push(**pCategoryId**) categoryIds.push(**categoryId**)

}

}

**const** title = (

<**span**>

<**LinkButton onClick=**{() => **this**.**props**.history.*goBack*()}>

<**Icon type='arrow-left' style=**{{**fontSize**: 20}}/>

</**LinkButton**>

### ———————————————————————————

{**isUpdate** ? **'**修改商品**'** : **'**添加商品**'**}

</**span**>

)

*//* 指定*form* 的*item* 布局的对象

**const** formItemLayout = { **labelCol**: { **span**: 2 }, **wrapperCol**: { **span**: 8 }

}

**return** (

<**Card title=**{title}>

<**Form**>

<**Item label="**商品名称**"** {...formItemLayout}>

{

getFieldDecorator(**'name'**, { **initialValue**: **product**.name, **rules**: [

{**required**: **true**, **message**: **'**商品名称必须输入**'**}

]

})(

<**Input placeholer='**请输入商品名称**'**/>

)

}

</**Item**>

<**Item label="**商品描述**"** {...formItemLayout}>

{

getFieldDecorator(**'desc'**, { **initialValue**: **product**.desc, **rules**: [

{**required**: **true**, **message**: **'**商品描述必须输入**'**}

]

})(

<**TextArea placeholder="**请输入商品描述**" autosize** />

)

}

</**Item**>

<**Item label="**商品价格**"** {...formItemLayout}>

{

getFieldDecorator(**'price'**, { **initialValue**: **product**.price, **rules**: [

{**required**: **true**, **message**: **'**商品价格必须输入**'**},

{**validator**: **this**.*validatePrice*}

]

})(

<Input type='number' placeholer='请输入商品价格' addonAfter='元'/>

)

}

</**Item**>

<**Item label="**商品分类**"** {...formItemLayout}>

{

getFieldDecorator(**'categoryIds'**, { **initialValue**: categoryIds, **rules**: [

{**required**: **true**, **message**: **'**商品分类必须输入**'**}

]

})(

<**Cascader options=**{**options**}

**loadData=**{**this**.*loadData*}

)

}

</**Item**>

<**Item label="**商品图片**"** {...formItemLayout}>

<**PicturesWall ref=**{**this**.**pw**} **imgs=**{**product**.imgs}/>

</**Item**>

<**Item**

**label="**商品详情**"**

**labelCol=**{{ **span**: 2 }}

**wrapperCol=**{{ **span**: 20 }}>

<**RichTextEditor ref=**{**this**.**editor**} **detail=**{**product**.detail}/>

</**Item**>

<**Button type='primary' onClick=**{**this**.*submit*}>提交</**Button**>

</**Form**>

/>

</**Card**>

)

}

}

**export default** Form.*create*()(ProductAddUpdate)

## product/pictures-wall.jsx

**import** React **from 'react'**

**import** PropTypes **from 'prop-types'**

**import** {Upload, Icon, Modal, message} **from 'antd'**

**import** {BASE\_IMG\_PATH, UPLOAD\_IMG\_NAME} **from '../../utils/constants' import** {*reqDeleteImg*} **from '../../api'**

*/\**

管理商品图片的组件*(*上传*/*删除图片*)*

*\*/*

**export default class** PicturesWall **extends** React.Component {

**static** *propTypes* = {

**imgs**: PropTypes.array

}

constructor (props) {

**super**(props)

**let** fileList = []

*//* 如果传入了*imgs,* 生成一个对应的*fileList*

**const** imgs = **this**.**props**.**imgs if** (imgs && imgs.**length** > 0) {

fileList = imgs.map((img, index) => ({

**uid**: -index,

**name**: img,

**status**: **'done'**, *// loading:* 上传中*, done:* 上传完成*, remove:* 删除

**url**: BASE\_IMG\_PATH + img,

}))

}

*//*初始化状态

**this**.**state** = {

**previewVisible**: **false**, *//* 是否显示大图预览

**previewImage**: **''**, *//* 大图的*url*

**fileList**: fileList *//* 所有需要显示的图片信息对象的数组

}

}

*/\**

得到当前已上传的图片文件名的数组

*\*/*

*getImgs* = () => **this**.**state**.**fileList**.map(file => file.**name**)

*/\**

关闭大图预览

*\*/*

*handleCancel* = () => **this**.setState({**previewVisible**: **false**})

*/\**

预览大图

*\*/*

*handlePreview* = (file) => {

**this**.setState({

**previewImage**: file.**url** || file.thumbUrl, *//* 需要显示的大图的*url*

**previewVisible**: **true**,

})

}

*/\**

*file:* 当前操作文件信息对象

*fileList:* 所有文件信息对象的数组

*\*/*

*handleChange* = **async** ({file, fileList}) => {

***console***.log(**'handleChange()'**, file, fileList)

*//* 如果上传图片完成

**if** (file.**status** === **'done'**) { **const** result = file.**response if** (result.**status** === 0) {

message.success(**'**上传成功了**'**)

**const** {name, url} = result.**data**

file = fileList[fileList.**length** - 1] file.**name** = name

file.**url** = url

} **else** {

message.error(**'**上传失败了**'**)

}

} **else if** (file.**status** === **'removed'**) { *//* 删除图片

**const** result = **await** *reqDeleteImg*(file.**name**)

**if**(result.**status**===0) { message.success(**'**删除图片成功**'**)

} **else** {

message.error(**'**删除图片失败**'**)

}

}

*//* 更新*fileList* 状态

**this**.setState({fileList})

}

render() {

**const** {**previewVisible**, **previewImage**, **fileList**} = **this**.**state**

**const** uploadButton = (

<**div**>

<**Icon type="plus"**/>

<**div**>上传图片</**div**>

</**div**>

)

**return** (

<**div**>

<**Upload**

**action="/manage/img/upload" accept="image/\*"**

**name=** {UPLOAD\_IMG\_NAME}

**listType="picture-card" fileList=**{**fileList**}

**onPreview=**{**this**.*handlePreview*}

**onChange=**{**this**.*handleChange*}

>

{uploadButton}

</**Upload**>

<**Modal visible=**{**previewVisible**} **footer=**{**null**} **onCancel=**{**this**.*handleCancel*}>

<**img alt="example" style=**{{**width**: **'100%'**}} **src=**{**previewImage**}/>

</**Modal**>

</**div**>

)

}

}

## product/rich-text-editor.jsx

**import** React, {Component} **from 'react' import** PropTypes **from 'prop-types'**

**import** {Editor} **from 'react-draft-wysiwyg'**

**import** {EditorState, *convertToRaw*, ContentState} **from 'draft-js' import** draftToHtml **from 'draftjs-to-html'**

**import** htmlToDraft **from 'html-to-draftjs'**

**import 'react-draft-wysiwyg/dist/react-draft-wysiwyg.css'**

*/\**

用来指定商品详情信息的富文本编程器组件

*\*/*

**export default class** RichTextEditor **extends** Component {

**static** *propTypes* = {

**detail**: PropTypes.string

}

constructor (props) {

**super**(props)

*//* 根据传入的*html* 文本初始显示

**const** detail = **this**.**props**.**detail let** editorState

**if**(detail) { *//* 如果传入才需要做处理

**const** blocksFromHtml = htmlToDraft(detail)

**const** { contentBlocks, entityMap } = blocksFromHtml

**const** contentState = ContentState.createFromBlockArray(contentBlocks, entityMap) editorState = EditorState.createWithContent(contentState)

} **else** {

editorState = EditorState.createEmpty()

}

*//* 初始化状态

**this**.**state** = { editorState

}

}

*/\**

当输入改变时立即保存状态数据

*\*/*

*onEditorStateChange* = (editorState) => {

**this**.setState({ editorState,

})

}

*/\**

得到输入的富文本数据

*\*/*

*getDetail* = () => {

**return** draftToHtml(*convertToRaw*(**this**.**state**.**editorState**.getCurrentContent()))

}

render() {

**const** {**editorState**} = **this**.**state**

**return** (

<**Editor editorState=**{**editorState**}

**editorStyle=**{{**height**: 250, **border**: **'1px solid #000'**, **padding**: **'0 30px'**}}

**onEditorStateChange=**{**this**.*onEditorStateChange*}

)

}

}

/>

* + 1. **product/product.less**

.product-detail {

.**left** {

**margin-right**: 10**px**; **font-size**: 18**px**; **font-weight**: **bold**;

}

}

* + 1. **utils/constants.js**

*/\**

包含*n* 个应用中的常量字符串模块

*\*/*

**export const** PAGE\_SIZE = 5 *//* 每页显示条目数

**export const** BASE\_IMG\_PATH = **'http://localhost:5000/upload/'** *//* 上传的图片的基础地址

**export const** UPLOAD\_IMG\_NAME = **'image'** *//* 上传图片的参数名

* 1. 角色管理
     1. **api/index.js:**

*//* 添加角色

**export const** *reqAddRole* = (roleName) => *ajax*(**'/manage/role/add'**, {roleName}, **'POST'**)

*//* 获取角色列表

**export const** *reqRoles* = () => *ajax*(**'/manage/role/list'**)

*//* 更新角色*(*给角色设置权限*)*

**export const** *reqUpdateRole* = (role) => *ajax*(**'/manage/role/update'**, role, **'POST'**)

* + 1. **add-form.jsx**

**import** React, {PureComponent} **from 'react' import** PropTypes **from 'prop-types'**

**import** { Form, Input,

} from 'antd'

*/\**

用来添加角色的*form* 组件

*\*/*

**class** AddForm **extends** PureComponent {

**static** *propTypes* = {

**setForm**: PropTypes.*func*.isRequired

}

componentWillMount() {

**this**.**props**.**setForm**(**this**.**props**.**form**)

}

render() {

**const** {getFieldDecorator} = **this**.**props**.**form const** formItemLayout = {

**labelCol**: {**span**: 5},

**wrapperCol**: {**span**: 16}

}

**return** (

<**Form**>

<**Form.Item label="**角色名称**"** {...formItemLayout}>

{

getFieldDecorator(**'roleName'**, {

**initialValue**: **''**

})(

<**Input type="text" placeholder="**请输入角色名称**"**/>

)

}

</**Form.Item**>

</**Form**>

)

}

}

**export default** AddForm = Form.*create*()(AddForm)

* + 1. **auth-form.jsx**

**import** React, {PureComponent} **from 'react' import** PropTypes **from 'prop-types'**

import {

### ———————————————————————————

Form,

Input, Tree

} from 'antd'

**import** menuList **from '../../config/menuConfig'**

**const** Item = Form.*Item*

**const** { TreeNode } = Tree;

*/\**

添加分类的*form* 组件

*\*/*

**export default class** AuthForm **extends** PureComponent {

**static** *propTypes* = {

**role**: PropTypes.object

}

constructor (props) {

**super**(props)

*//* 根据传入角色的*menus* 生成初始状态

**const** {menus} = **this**.**props**.role **this**.**state** = {

**checkedKeys**: menus

}

}

*/\**

为父组件提交获取最新*menus* 数据的方法

*\*/*

*getMenus* = () => **this**.**state**.checkedKeys

*getTreeNodes* = (menuList) => {

**return** menuList.reduce((pre, item) => { pre.push(

<**TreeNode title=**{item.**title**} **key=**{item.**key**}>

{item.**children** ? **this**.*getTreeNodes*(item.**children**) : **null**}

</**TreeNode**>

)

### ———————————————————————————

**return** pre

}, [])

}

*//* 选中某个*node* 时的回调

*onCheck* = checkedKeys => { ***console***.log(**'onCheck'**, checkedKeys); **this**.setState({ checkedKeys });

};

componentWillMount () {

**this**.**treeNodes** = **this**.*getTreeNodes*(menuList)

}

*//* 根据新传入的*role* 来更新*checkedKeys* 状态

*/\**

当组件接收到新的属性时自动调用

*\*/*

componentWillReceiveProps (nextProps) { ***console***.log(**'componentWillReceiveProps()'**, nextProps) **const** menus = nextProps.role.**menus**

**this**.setState({ **checkedKeys**: menus

})

*// this.state.checkedKeys = menus*

}

render() {

***console***.log(**'AuthForm render()'**) **const** {role} = **this**.**props**

**const** {checkedKeys} = **this**.**state**

*//* 指定*Item* 布局的配置对象

**const** formItemLayout = {

**labelCol**: { **span**: 4 }, *//* 左侧*label* 的宽度

**wrapperCol**: { **span**: 15 }, *//* 右侧包裹的宽度

}

return (

<**div**>

<**Item label='**角色名称**'** {...formItemLayout}>

<**Input value=**{role.**name**} **disabled**/>

### ———————————————————————————

</**Item**>

<**Tree**

**defaultExpandAll=**{**true**}

{checkedKeys}

**s**.*onCheck*}

<**TreeNode title="**平台权限**" key="all"**>

{**this**.**treeNodes**}

</**TreeNode**>

</**Tree**>

</**div**>

)

}

}

**checkable**

|  |  |  |
| --- | --- | --- |
| **checkedKeys=** | | |
| **onCheck=** | | {**thi** |
| > |  | |

* + 1. **role.jsx**

**import** React, {PureComponent} **from 'react' import** {

Card, Button, Table, Modal, message

} from 'antd'

**import** {PAGE\_SIZE} **from "../../utils/constants"**

**import** {*reqRoles*, *reqAddRole*, *reqUpdateRole*} **from '../../api' import** AddForm **from './add-form'**

**import** AuthForm **from './auth-form'**

**import** memoryUtils **from "../../utils/memoryUtils" import** {*formateDate*} **from '../../utils/dateUtils'**

*/\**

角色路由

*\*/*

———————————————————————————

**export default class** Role **extends** PureComponent {

**state** = {

**roles**: [], *//* 所有角色的列表

**role**: {}, *//* 选中的*role*

**isShowAdd**: **false**, *//* 是否显示添加界面

**isShowAuth**: **false**, *//* 是否显示设置权限界面

}

constructor (props) {

**super**(props)

**this**.**auth** = React.createRef()

}

*initColumn* = () => {

**this**.**columns** = [

{

**title**: **'**角色名称**'**, **dataIndex**: **'name'**

},

{

**title**: **'**创建时间**'**, **dataIndex**: **'create\_time'**,

render: (create\_time) => *formateDate*(create\_time)

},

{

**title**: **'**授权时间**'**, **dataIndex**: **'auth\_time'**,

**render**: *formateDate*

},

{

**title**: **'**授权人**'**, **dataIndex**: **'auth\_name'**

},

]

}

*getRoles* = **async** () => {

**const** result = **await** *reqRoles*()

**if** (result.**status**===0) {

**const** roles = result.**data**

### ———————————————————————————

**this**.setState({

roles

})

}

}

*onRow* = (role) => {

return {

onClick: event => { *//* 点击行

***console***.log(**'row onClick()'**, role)

*// alert('*点击行*')*

**this**.setState({ role

})

},

}

}

*/\**

添加角色

*\*/*

*addRole* = () => {

*//* 进行表单验证*,* 只能通过了才向下处理

**this**.**form**.validateFields(**async** (error, values) => {

**if** (!error) {

*//* 隐藏确认框

**this**.setState({ **isShowAdd**: **false**

})

*//* 收集输入数据

**const** {roleName} = values

**this**.**form**.resetFields()

*//* 请求添加

**const** result = **await** *reqAddRole*(roleName)

*//* 根据结果提示*/*更新列表显示

**if** (result.**status**===0) { message.success(**'**添加角色成功**'**)

*// this.getRoles()*

### ———————————————————————————

*//* 新产生的角色

**const** role = result.**data**

*//* 更新*roles* 状态

*/\*const roles = this.state.roles roles.push(role)*

*this.setState({ roles*

*})\*/*

*//* 更新*roles* 状态*:* 基于原本状态数据更新

**this**.setState(state => ({

**roles**: [...state.roles, role]

}))

} **else** {

message.success(**'**添加角色失败**'**)

}

}

})

}

*/\**

更新角色

*\*/*

*updateRole* = **async** () => {

*//* 隐藏确认框

**this**.setState({ **isShowAuth**: **false**

})

**const** role = **this**.**state**.**role**

*//* 得到最新的 *menus*

**const** menus = **this**.**auth**.**current**.*getMenus*() role.**menus** = menus

role.**auth\_time** = Date.now()

role.**auth\_name** = memoryUtils.**user**.**username**

*//* 请求更新

### ———————————————————————————

**const** result = **await** *reqUpdateRole*(role)

**if** (result.**status**===0) { message.success(**'**设置角色权限成功**'**)

*// this.getRoles()*

**this**.setState({

**roles**: [...**this**.**state**.**roles**]

})

}

}

componentWillMount () {

**this**.*initColumn*()

}

componentDidMount () {

**this**.*getRoles*()

}

render() {

***console***.log(**'Role render()'**)

**const** {**roles**, role, **isShowAdd**, **isShowAuth**} = **this**.**state**

**const** title = (

<**span**>

<**Button type='primary' onClick=**{() => **this**.setState({**isShowAdd**: **true**})}>创建角色</**Button**> **&nbsp;&nbsp;**

<**Button type='primary' disabled=**{!role.**\_id**} **onClick=**{() =>

**this**.setState({**isShowAuth**: **true**})}>设置角色权限</**Button**>

</**span**>

)

**return** (

<**Card title=**{title}>

<**Table**

**bordered**

rowKey='\_id'

{**roles**}

|  |  |
| --- | --- |
| **dataSource=** | |
| **columns=** | {**th** |
| **pagination=** | |

is.columns}

{{**defaultPageSize**: PAGE\_SIZE}}

**rowSelection=**{{**type**: **'radio'**, **selectedRowKeys**: [role.**\_id**]}}

{**this**.*onRow*}

/>

onRow=

### ———————————————————————————

**title="**添加角色**"**

**visible=**{**isShowAdd**} **onOk=**{**this**.*addRole*}

**onCancel=**{() => {

**this**.setState({**isShowAdd**: **false**}) **this**.**form**.resetFields()

}}

>

<**AddForm**

**setForm=**{(form) => **this**.**form** = form}

</**Modal**>

<**Modal**

**title="**设置角色权限**"**

**visible=**{**isShowAuth**} **onOk=**{**this**.*updateRole*}

**onCancel=**{() => {

**this**.setState({**isShowAuth**: **false**})

}}

>

<**AuthForm ref=**{**this**.**auth**} **role=**{role}/>

</**Modal**>

</**Card**>

)

}

}

<**Modal**

/>

* 1. 用户管理
     1. **api/index.js**

*//* 添加*/*更新用户

**export const** *reqAddOrUpdateUser* = (user) => *ajax*(**'/manage/user/'**+(user.**\_id** ? **'update'** :

**'add'**), user, **'POST'**)

———————————————————————————

*//* 获取用户列表

**export const** *reqUsers* = () => *ajax*(**'/manage/user/list'**)

*//* 删除用户

**export const** *reqDeleteUser* = (userId) => *ajax*(**'/manage/user/delete'**, {userId}, **'POST'**)

* + 1. **user-form.jsx**

**import** React, {Component} **from 'react' import** PropTypes **from 'prop-types' import** {

Form, Input, Select,

} **from 'antd'**

**const** FormItem = Form.*Item*

**const** Option = Select.***Option***

*/\**

用来添加或更新的*form* 组件

*\*/*

**class** UserForm **extends** Component {

**static** *propTypes* = {

**setForm**: PropTypes.func.isRequired,

**user**: PropTypes.object,

**roles**: PropTypes.array

}

componentWillMount() {

**this**.**props**.**setForm**(**this**.**props**.**form**)

}

render() {

**const** {getFieldDecorator} = **this**.**props**.**form const** formItemLayout = {

**labelCol**: {**span**: 4},

**wrapperCol**: {**span**: 16}

———————————————————————————

}

**const** {**user**, **roles**} = **this**.**props return** (

<**Form** {...formItemLayout}>

<**FormItem label="**用户名**"**>

{

getFieldDecorator(**'username'**, {

initialValue: user.username

})(

<**Input type="text" placeholder="**请输入用户名**"**/>

)

}

</**FormItem**>

{

!**user**.**\_id** ?

(

<**FormItem label="**密码**"**>

{

getFieldDecorator(**'password'**, {

**initialValue**: **''**

})(

<**Input type="password" placeholder="**请输入密码**"**/>

)

}

</**FormItem**>

) : **null**

}

<**FormItem label="**手机号**"**>

{

getFieldDecorator(**'phone'**, {

initialValue: user.phone

})(

<**Input type="phone" placeholder="**请输入手机号**"**/>

)

}

</**FormItem**>

### ———————————————————————————

<**FormItem label="**邮箱**"**>

{

getFieldDecorator(**'email'**, {

**initialValue**: **user**.**email**

})(

<**Input type="email" placeholder="**请输入邮箱**"**/>

)

}

</**FormItem**>

<**FormItem label="**角色**"**>

{

getFieldDecorator(**'role\_id'**, {

**initialValue**: **user**.role\_id

})(

<**Select style=**{{**width**: 200}} **placeholder='**请选择角色**'**>

{

**roles**.map(role => <**Option key=**{role.**\_id**} **value=**{role.**\_id**}>{role.**name**}</**Option**>)

}

</**Select**>

)

}

</**FormItem**>

</**Form**>

)

}

}

**export default** Form.*create*()(UserForm)

* + 1. **user.jsx**

**import** React, {Component} **from 'react' import** {

Card, Button, Table,

Modal,

———————————————————————————

} from 'antd'

**import** *LinkButton* **from '../../components/link-button' import** UserForm **from './user-form'**

import {

*reqUsers*, *reqAddOrUpdateUser*, *reqDeleteUser*

} from '../../api'

**import** {*formateDate*} **from '../../utils/dateUtils'**

**import** {PAGE\_SIZE} **from "../../utils/constants"**;

*/\**

后台管理的用户管理路由组件

*\*/*

**export default class** User **extends** Component {

**state** = {

**isShow**: **false**, *//* 是否显示对话框 **users**: [], *//* 所有用户的列表 **roles**: [], *//* 所有角色的列表

}

*/\**

初始化*Table* 的字段列表

*\*/*

*initColumns* = () => {

**this**.**columns** = [

{

**title**: **'**用户名**'**, **dataIndex**: **'username'**

},

{

**title**: **'**邮箱**'**,

dataIndex: 'email'

},

{

**title**: **'**电话**'**, **dataIndex**: **'phone'**

},

{

**title**: **'**注册时间**'**,

### ———————————————————————————

dataIndex: 'create\_time',

**render**: *formateDate*

},

{

**title**: **'**所属角色**'**, **dataIndex**: **'role\_id'**,

render: value => **this**.**roleNames**[value]

},

{

**title**: **'**操作**'**,

render: (user) => (

<**span**>

<**LinkButton onClick=**{() => **this**.*showUpdate*(user)}>修改</**LinkButton**> **&nbsp;&nbsp;**

<**LinkButton onClick=**{() => **this**.*clickDelete*(user)}>删除</**LinkButton**>

</**span**>

)

},

]

}

*/\**

根据角色的数组生成一个包含所有角色名的对象容器

*\*/*

*initRoleNames* = (roles) => {

**this**.**roleNames** = roles.reduce((pre, role) => { pre[role.**\_id**] = role.**name**

**return** pre

}, {})

}

*/\**

响应点击删除用户

*\*/*

*clickDelete* = (user) => { Modal.*confirm*({

**content**: **`**确定删除${user.**username**}吗**?`**,

onOk: **async** () => {

**const** result = **await** *reqDeleteUser*(user.**\_id**)

**if** (result.**status** === 0) {

**this**.*getUsers*()

}

### ———————————————————————————

}

})

}

*/\**

显示修改用户的界面

*\*/*

*showUpdate* = (user) => {

*//* 保存 *user*

**this**.**user** = user

**this**.setState({ **isShow**: **true**

})

}

*/\**

异步获取所有用户列表

*\*/*

*getUsers* = **async** () => {

**const** result = **await** *reqUsers*()

**if** (result.**status** === 0) {

**const** {users, roles} = result.**data**

*//* 初始化生成一个包含所有角色名的对象容器 *{\_id1: name1, \_id2: nam2}*

**this**.*initRoleNames*(roles) **this**.setState({

users, roles

})

}

}

*/\**

显示添加用户的界面

*\*/*

*showAddUser* = () => { **this**.**user** = **null this**.setState({

**isShow**: **true**

})

}

### ———————————————————————————

*/\**

添加*/*更新用户

*\*/*

*AddOrUpdateUser* = **async** () => {

*//* 获取表单数据

**const** user = **this**.**form**.getFieldsValue() **this**.**form**.resetFields()

**if** (**this**.**user**) {

user.**\_id** = **this**.**user**.**\_id**

}

**this**.setState({ **isShow**: **false**

})

**const** result = **await** *reqAddOrUpdateUser*(user)

**if** (result.**status** === 0) {

**this**.*getUsers*()

}

}

componentWillMount() {

**this**.*initColumns*()

}

componentDidMount() {

**this**.*getUsers*()

}

render() {

**const** {**users**, **roles**, **isShow**} = **this**.**state**

**const** user = **this**.**user** || {}

**const** title = <**Button type="primary" onClick=**{**this**.*showAddUser*}>创建用户</**Button**>

return (

<**div**>

<**Card title=**{title}>

<Table

columns={this.columns}

rowKey='\_id'

### ———————————————————————————

**dataSource=**{**users**}

{{**defaultPageSize**: PAGE\_SIZE, **showQuickJumper**: **true**}}

<**Modal**

ser.**\_id** ? **'**修改用户**'** : **'**添加用户**'**}

{**isShow**}

**onCancel=**{() => **this**.setState({**isShow**: **false**})} **onOk=**{**this**.*AddOrUpdateUser*}

>

<**UserForm**

**setForm=**{(form) => **this**.**form** = form}

**user=**{user}

**roles=**{**roles**}

</**Modal**>

</**Card**>

</**div**>

)

}

}

/>

**pagination=**

**bordered**

/>

|  |  |
| --- | --- |
| **title=** | {u |
| **visible=** | |

* + 1. 导航权限控制

1. **left-nav.jsx**

*/\**

判断当前用户是否有看到当前*item* 对应菜单项的权限

*\*/*

*hasAuth* = (item) => {

**const** key = item.**key**

**const** menuSet = **this**.**menuSet**

*/\**

1. 如果菜单项标识为公开
2. 如果当前用户是*admin*
3. 如果菜单项的*key* 在用户的*menus* 中

———————————————————————————

*\*/*

**if**(item.**isPublic** || memoryUtils.*user*.username===**'admin'** || menuSet.has(key)) {

return true

*// 4.* 如果有子节点*,* 需要判断有没有一个*child* 的*key* 在*menus* 中

} **else if**(item.**children**){

**return** !!item.**children**.*find*(child => menuSet.has(child.**key**))

}

}

*/\**

返回包含*n* 个*<Item>*和*<SubMenu>*的数组

*\*/*

*getMenuNodes* = (list) => {

*//* 得到当前请求的*path*

**const** path = **this**.**props**.**location**.**pathname**

**return** list.reduce((pre, item) => {

**if** (**this**.*hasAuth*(item)) {

**if** (!item.**children**) { pre.push((

<**Menu.Item key=**{item.**key**}>

<**Link to=**{item.**key**}>

<**Icon type=**{item.**icon**}/>

<**span**>{item.**title**}</**span**>

</**Link**>

</**Menu.Item**>

))

} **else** { pre.push((

<**SubMenu key=**{item.**key**} **title=**{<**span**><**Icon type=**{item.**icon**}/><**span**>{item.**title**}</**span**></**span**>}>

{

**this**.*getMenuNodes*(item.**children**)

}

</**SubMenu**>

))

**if**(item.**children**.*find*(cItem => path.indexOf(cItem.**key**)===0)) {

**this**.**openKey** = item.**key**

}

}

## config/menuConfig.js

}

**return** pre

}, [])

}

componentWillMount () {

**this**.**menuSet** = **new** Set(memoryUtils.*user*.**role**.**menus** || [])

**this**.**menuNodes** = **this**.*getMenuNodes*(menuList)

}

{

**title**: **'**首页**'**, *//* 菜单标题名称 **key**: **'/home'**, *//* 对应的*path* **icon**: **'home'**, *//* 图标名称 isPublic: true, *//* 开放的

}

* 1. 使用 **redux** 管理状态

### 下载依赖

**yarn add redux react-redux redux-thunk redux-devtools-extension**

* + 1. **redux/store.js**

*/\**

*redux* 最核心的管理对象*: store*

*\*/*

**import** {createStore, *applyMiddleware*} **from 'redux' import** thunk **from 'redux-thunk'**

import {composeWithDevTools} from 'redux-devtools-extension'

**import** reducer **from './reducer'**

*//* 向外暴露*store* 对象

* + 1. **redux/reducer.js**

**export default** createStore(reducer, **composeWithDevTools**(*applyMiddleware*(thunk))) *//* 创建*store* 对象内部会第一次调用*reducer()*得到初始状态值

*/\**

*reducer* 函数模块*:* 根据当前*state* 和指定*action* 返回一个新的*state*

*\*/*

**import** {*combineReducers*} **from 'redux'**

**import** { SET\_HEAD\_TITLE, RECEIVE\_USER, SHOW\_ERROR\_MSG, RESET\_USER

} **from './action-types'**

**import** storageUtils **from "../utils/storageUtils"**;

*/\**

管理*headTitle* 状态数据的*reducer*

*\*/*

**const** initHeadTitle = **'**首页**'**

**function** *headTitle*(state = initHeadTitle, action) {

***console***.log(**'headTitle()'**, state, action)

**switch** (action.**type**) {

**case** SET\_HEAD\_TITLE:

**return** action.**data default**:

**return** state

}

}

*/\**

管理*user* 状态数据的*reducer*

*\*/*

**const** initUser = storageUtils.getUser()

———————————————————————————

**function** *user*(state = initUser, action) { ***console***.log(**'user()'**, state, action) **switch** (action.**type**) {

**case** RECEIVE\_USER:

**return** action.**user case** SHOW\_ERROR\_MSG:

**return** {...state, **errorMsg**: action.**errorMsg**}

**case** RESET\_USER:

**return** {}

default:

**return** state

}

}

*/\**

向外暴露合并后产生的总*reducer* 函数总的*state* 的结构*:*

*{*

*headerTitle: '', user: {}*

*}*

*\*/*

**export default** *combineReducers*({ *headTitle*,

*user*,

})

* + 1. **redux/actions.js**

*/\**

包含*n* 个用来创建*action* 的工厂函数*(action creator)*

*\*/*

**import** { SET\_HEAD\_TITLE, RECEIVE\_USER, SHOW\_ERROR\_MSG,

———————————————————————————

RESET\_USER

} from './action-types'

**import** {*reqLogin*} **from '../api'**

**import** storageUtils **from "../utils/storageUtils"**;

*/\**

设置头部标题的同步*action*

*\*/*

**export const** *setHeadTitle* = (headTitle) => ({**type**: SET\_HEAD\_TITLE, **data**: headTitle})

*/\**

接收用户的同步*action*

*\*/*

**export const** *receiveUser* = (user) => ({**type**: RECEIVE\_USER, user})

*/\**

显示错误信息的同步*action*

*\*/*

**export const** *showErrorMsg* = (errorMsg) => ({**type**: SHOW\_ERROR\_MSG, errorMsg})

*/\**

退出登陆的同步*action*

*\*/*

**export const** *logout* = () => { storageUtils.removeUser() **return** {**type**: RESET\_USER}

}

*/\**

登陆的异步*action*

*\*/*

**export const** *login* = (username, password) => {

**return async** dispatch => {

**const** result = **await** *reqLogin*(username, password)

**if** (result.**status** === 0) { **const** user = result.**data** storageUtils.saveUser(user) dispatch(*receiveUser*(user))

} **else** {

**const** msg = result.msg dispatch(*showErrorMsg*(msg))

## redux/action-types.js

}

}

}

*/\**

包含*n* 个*action type* 常量名称的模块

*\*/*

**export const** SET\_HEAD\_TITLE = **'set\_head\_title'** *//* 设置头部标题

**export const** RECEIVE\_USER = **'receive\_user'** *//* 接收新的用户

**export const** SHOW\_ERROR\_MSG = **'show\_error\_msg'** *//* 显示错误提示信息

**export const** RESET\_USER = **'reset\_user'** *//* 重置用户

* + 1. **index.js**

*/\**

入口*js*

*\*/*

**import** React **from 'react'**

import ReactDOM from 'react-dom' import {Provider} from 'react-redux'

import store from './redux/store' import App from './App'

ReactDOM.render((

<**Provider store=**{store}>

<**App** />

* + 1. **components/left-nav/left-nav.js**

</**Provider**>

), **document**.getElementById(**'root'**))

**import** React, {Component} **from 'react' import** PropTypes **from 'prop-types'**

**import** {Link, withRouter} **from 'react-router-dom' import** {Menu, Icon} **from 'antd'**

**import** {**connect**} **from 'react-redux'**

**import** {*setHeadTitle*} **from '../../redux/actions' import** logo **from '../../assets/images/logo.png' import** menuList **from '../../config/menuConfig' import './index.less'**

**const** SubMenu = Menu.*SubMenu*;

*/\**

左侧导航的组件

*\*/*

**class** LeftNav **extends** Component {

*/\**

判断当前登陆用户对*item* 是否有权限

*\*/*

*hasAuth* = (item) => {

**const** {key, isPublic} = item

**const** menus = **this**.**props**.user.**role**.**menus const** username = **this**.**props**.user.**username**

*/\**

1. 如果当前用户是*admin*
2. 如果当前*item* 是公开的
3. 当前用户有此*item* 的权限*: key* 有没有*menus* 中

*\*/*

**if** (username === **'admin'** || isPublic || menus.indexOf(key) !== -1) {

**return true**

### ———————————————————————————

} **else if** (item.**children**) { *// 4.* 如果当前用户有此*item* 的某个子*item* 的权限

**return** !!item.**children**.find(child => menus.indexOf(child.**key**) !== -1)

}

return false

}

*/\**

根据*menu* 的数据数组生成对应的标签数组

使用*map() +* 递归调用

*\*/*

*getMenuNodes\_map* = (menuList) => {

**return** menuList.map(item => {

**if** (!item.**children**) {

**return** (

<**Menu.Item key=**{item.**key**}>

<**Link to=**{item.**key**}>

<**Icon type=**{item.**icon**}/>

<**span**>{item.**title**}</**span**>

</**Link**>

</**Menu.Item**>

)

} **else** {

**return** (

<**SubMenu key=**{item.**key**}

**title=**{

<**span**>

<**Icon type=**{item.**icon**}/>

<**span**>{item.**title**}</**span**>

</**span**>

}

>

{**this**.*getMenuNodes*(item.**children**)}

</**SubMenu**>

)

}

})

}

*/\**

### ———————————————————————————

根据*menu* 的数据数组生成对应的标签数组

使用 *reduce() +* 递归调用

*\*/*

*getMenuNodes* = (menuList) => {

*//* 得到当前请求的路由路径

**const** path = **this**.**props**.location.**pathname**

*//* 如果当前请求的是根路径*,* 设置头部标题为首页

**if**(path===**'/'**) {

**this**.**props**.setHeadTitle(**'**首页**'**)

}

**return** menuList.reduce((pre, item) => {

*//* 如果当前用户有*item* 对应的权限*,* 才需要显示对应的菜单项

**if** (**this**.*hasAuth*(item)) {

*//* 向*pre* 添加*<Menu.Item>*

**if** (!item.**children**) {

*//* 一旦请求路径匹配上当前*item,* 将*item* 的*title* 保存到*redux*

**if**(item.**key**===path || path.indexOf(item.**key**)===0) {

**this**.**props**.setHeadTitle(item.**title**)

}

pre.push((

<**Menu.Item key=**{item.**key**}>

<**Link to=**{item.**key**} **onClick=**{() => **this**.**props**.setHeadTitle(item.**title**)}>

<**Icon type=**{item.**icon**}/>

<**span**>{item.**title**}</**span**>

</**Link**>

</**Menu.Item**>

))

} **else** {

*//* 查找一个与当前请求路径匹配的子*Item*

**const** cItem = item.**children**.find(cItem => path.indexOf(cItem.**key**) === 0)

*//* 如果存在*,* 说明当前*item* 的子列表需要打开

**if** (cItem) {

**this**.**openKey** = item.**key**

}

*//* 向 *pre* 添加*<SubMenu>*

pre.push((

### ———————————————————————————

<SubMenu

**key=**{item.**key**}

**title=**{

<**span**>

<**Icon type=**{item.**icon**}/>

<**span**>{item.**title**}</**span**>

</**span**>

}

{**this**.*getMenuNodes*(item.**children**)}

</**SubMenu**>

))

}

}

**return** pre

}, [])

}

*/\**

在第一次*render()*之前执行一次

为第一个*render()*准备数据*(*必须同步的*)*

*\*/*

componentWillMount() {

**this**.**menuNodes** = **this**.*getMenuNodes*(menuList)

}

render() {

*//* 得到当前请求的路由路径

**let** path = **this**.**props**.location.**pathname**

***console***.log(**'render()'**, path)

**if** (path.indexOf(**'/product'**) === 0) { *//* 当前请求的是商品或其子路由界面

>

path = **'/product'**

}

*//* 得到需要打开菜单项的 *key*

**const** openKey = **this**.**openKey**

return (

<div className="left-nav">

### ———————————————————————————

<**Link to='/' className="left-nav-header"**>

<**img src=**{logo} **alt="logo"**/>

<**h1**>硅谷后台</**h1**>

</**Link**>

<**Menu**

**mode="inline"**

**theme="dark" selectedKeys=**{[path]}

**defaultOpenKeys=**{[openKey]}

{

**this**.**menuNodes**

}

</**Menu**>

</**div**>

)

}

}

*/\**

*withRouter* 高阶组件*:*

包装非路由组件*,* 返回一个新的组件

新的组件向非路由组件传递*3* 个属性*: history/location/match*

*\*/*

**export default connect**(

state => ({**user**: state.**user**}),

{*setHeadTitle*}

)(withRouter(LeftNav))

>

* + 1. **components/header/header.jsx**

**import** React, {Component} **from 'react' import** {withRouter} **from 'react-router-dom' import** { Modal} **from 'antd'**

import {connect} from 'react-redux'

———————————————————————————

**import** {*logout*} **from '../../redux/actions' import** *LinkButton* **from '../link-button' import** {*reqWeather*} **from '../../api'**

import menuList from '../../config/menuConfig' import {*formateDate*} from '../../utils/dateUtils' import './index.less'

*/\**

左侧导航的组件

*\*/*

**class** Header **extends** Component {

**state** = {

**currentTime**: *formateDate*(Date.now()), *//* 当前时间字符串

**dayPictureUrl**: **''**, *//* 天气图片*url*

**weather**: **''**, *//* 天气的文本

}

*getTime* = () => {

*//* 每隔*1s* 获取当前时间*,* 并更新状态数据*currentTime*

**this**.**intervalId** = *setInterval*(() => {

**const** currentTime = *formateDate*(Date.now())

**this**.setState({currentTime})

}, 1000)

}

*getWeather* = **async** () => {

*//* 调用接口请求异步获取数据

**const** {dayPictureUrl, weather} = **await** *reqWeather*(**'**北京**'**)

*//* 更新状态

**this**.setState({dayPictureUrl, weather})

}

*getTitle* = () => {

*//* 得到当前请求路径

**const** path = **this**.**props**.**location**.**pathname let** title

menuList.forEach(item => {

**if** (item.**key**===path) { *//* 如果当前*item* 对象的*key* 与*path* 一样*,item* 的*title* 就是需要显示的*title*

title = item.**title**

### ———————————————————————————

} **else if** (item.**children**) {

*//* 在所有子*item* 中查找匹配的

**const** cItem = item.**children**.find(cItem => path.indexOf(cItem.**key**)===0)

*//* 如果有值才说明有匹配的

**if**(cItem) {

*//* 取出它的*title*

title = cItem.**title**

}

}

})

**return** title

}

*/\**

退出登陆

*\*/*

*logout* = () => {

*//* 显示确认框

Modal.*confirm*({

**content**: **'**确定退出吗**?'**, onOk: () => {

**this**.**props**.**logout**()

}

})

}

*/\**

第一次*render()*之后执行一次

一般在此执行异步操作*:* 发*ajax* 请求*/*启动定时器

*\*/*

componentDidMount () {

*//* 获取当前的时间

**this**.*getTime*()

*//* 获取当前天气

**this**.*getWeather*()

}

*/\**

*//* 不能这么做*:* 不会更新显示

*componentWillMount () { this.title = this.getTitle()*

*}\*/*

### ———————————————————————————

*/\**

当前组件卸载之前调用

*\*/*

componentWillUnmount () {

*//* 清除定时器

*clearInterval*(**this**.**intervalId**)

}

render() {

**const** {**currentTime**, **dayPictureUrl**, **weather**} = **this**.**state**

**const** username = **this**.**props**.**user**.**username**

*//* 得到当前需要显示的*title*

**const** title = **this**.**props**.**headTitle return** (

<**div className="header"**>

<**div className="header-top"**>

<**span**>欢迎, {username}</**span**>

<**LinkButton onClick=**{**this**.*logout*}>退出</**LinkButton**>

</**div**>

<**div className="header-bottom"**>

<**div className="header-bottom-left"**>{title}</**div**>

<**div className="header-bottom-right"**>

<**span**>{**currentTime**}</**span**>

<**img src=**{**dayPictureUrl**} **alt="weather"**/>

<**span**>{**weather**}</**span**>

</**div**>

</**div**>

</**div**>

)

}

}

**export default connect**( state => ({

**user**: state.**user**,

**headTitle**: state.**headTitle**

}),

{*logout*}

### ———————————————————————————

)(withRouter(Header))

* + 1. **pages/admin/admin.jsx**

**import** React, {Component} **from 'react'**

**import** {Redirect, Route, Switch} **from 'react-router-dom' import** { Layout } **from 'antd'**

**import** {**connect**} **from 'react-redux'**

**import** LeftNav **from '../../components/left-nav' import** Header **from '../../components/header' import** Home **from '../home/home'**

**import** Category **from '../category/category' import** Product **from '../product/product' import** Role **from '../role/role'**

**import** User **from '../user/user' import** Bar **from '../charts/bar' import** Line **from '../charts/line' import** Pie **from '../charts/pie'**

**const** { Footer, Sider, Content } = Layout

*/\**

后台管理的路由组件

*\*/*

**class** Admin **extends** Component { render () {

**const** user = **this**.**props**.user

*//* 如果内存没有存储 *user ==>* 当前没有登陆

**if**(!user || !user.**\_id**) {

*//* 自动跳转到登陆*(*在*render()*中*)*

**return** <**Redirect to='/login'**/>

}

**return** (

<**Layout style=**{{**minHeight**: **'100%'**}}>

<**Sider**>

<**LeftNav**/>

———————————————————————————

</**Sider**>

<**Layout**>

<**Header**>Header</**Header**>

<**Content style=**{{**margin**: 20, **backgroundColor**: **'#fff'**}}>

<**Switch**>

<**Route path='/home' component=**{Home}/>

<**Route path='/category' component=**{Category}/>

<**Route path='/product' component=**{Product}/>

<**Route path='/role' component=**{Role}/>

<**Route path='/user' component=**{User}/>

<Route path='/charts/bar' component={Bar}/>

<Route path='/charts/line' component={Line}/>

<Route path='/charts/pie' component={Pie}/>

<Redirect to='/home'/>

</**Switch**>

</**Content**>

<**Footer style=**{{**textAlign**: **'center'**, **color**: **'#cccccc'**}}>推荐使用谷歌浏览器，可以获得更佳页面操作体验</**Footer**>

</**Layout**>

</**Layout**>

)

}

}

export default connect(

state => ({**user**: state.**user**})

)(Admin)

* + 1. **pages/login/login.jsx**

**import** React, {PureComponent} **from 'react' import** {Redirect} **from 'react-router-dom' import** {**connect**} **from 'react-redux'**

**import** { Form, Icon, Input,

Button,

———————————————————————————

message

} from 'antd'

import './login.less'

import logo from '../../assets/images/logo.png' import {*login*} from '../../redux/actions'

**const** Item = Form.*Item //* 不能写在*import* 之前

*/\**

登陆的路由组件

*\*/*

**class** Login **extends** PureComponent {

*handleSubmit* = (event) => {

*//* 阻止事件的默认行为

event.preventDefault()

*//* 对所有表单字段进行检验

**this**.**props**.form.validateFields(**async** (err, values) => {

*//* 检验成功

**if** (!err) {

*//* 请求登陆

**const** {username, password} = values

**this**.**props**.login(username, password)

}

});

*//* 得到*form* 对象

*// const form = this.props.form*

*// //* 获取表单项的输入数据

*// const values = form.getFieldsValue()*

*// console.log('handleSubmit()', values)*

}

*/\**

对密码进行自定义验证

*\*/*

*/\**

用户名*/*密码的的合法性要求

1. 必须输入
2. 必须大于等于*4* 位

### ———————————————————————————

1. 必须小于等于*12* 位
2. 必须是英文、数字或下划线组成

*\*/*

*validatePwd* = (rule, value, callback) => { ***console***.log(**'validatePwd()'**, rule, value) **if** (!value) {

callback(**'**密码必须输入**'**)

} **else if** (value.**length** < 4) { callback(**'**密码长度不能小于 **4** 位**'**)

} **else if** (value.**length** > 12) {

callback(**'**密码长度不能大于 **12** 位**'**)

} **else if** (!/^[a-zA-Z0-9\_]+$/.test(value)) { callback(**'**密码必须是英文、数字或下划线组成**'**)

} **else** {

callback() *//* 验证通过

}

*// callback('xxxx') //* 验证失败*,* 并指定提示的文本

}

render() {

*//* 如果用户已经登陆*,* 自动跳转到管理界面

**const** user = **this**.**props**.user **if** (user && user.**\_id**) {

**return** <**Redirect to='/home'**/>

}

*//* 得到具强大功能的*form* 对象

**const** form = **this**.**props**.form

**const** {getFieldDecorator} = form;

**return** (

<**div className="login"**>

<**header className="login-header"**>

<**img src=**{logo} **alt="logo"**/>

<**h1**>React 项目: 后台管理系统</**h1**>

</**header**>

<section className="login-content">

<div className={user.errorMsg ? 'error-msg show' :

**'error-msg'**}>{user.**errorMsg**}</**div**>

<**h2**>用户登陆</**h2**>

<**Form onSubmit=**{**this**.*handleSubmit*} **className="login-form"**>

### ———————————————————————————

<**Item**>

{

*/\**

用户名*/*密码的的合法性要求

* 1. 必须输入
  2. 必须大于等于*4* 位
  3. 必须小于等于*12* 位
  4. 必须是英文、数字或下划线组成

*\*/*

}

{

getFieldDecorator(**'username'**, { *//* 配置对象*:* 属性名是特定的一些名称

*//* 声明式验证*:* 直接使用别人定义好的验证规则进行验证

**rules**: [

{**required**: **true**, **whitespace**: **true**, **message**: **'**用户名必须输入**'**},

{**min**: 4, **message**: **'**用户名至少 **4** 位**'**},

{**max**: 12, **message**: **'**用户名最多 **12** 位**'**},

{**pattern**: /^[a-zA-Z0-9\_]+$/, **message**: **'**用户名必须是英文、数字或下划线

组成**'**},

],

**initialValue**: **'admin'**, *//* 初始值

})(

<**Input**

**prefix=**{<**Icon type="user" style=**{{**color**: **'rgba(0,0,0,.25)'**}}/>}

)

}

</**Item**>

<**Form.Item**>

{

getFieldDecorator(**'password'**, {

**rules**: [

{

/>

**placeholder="**用户名**"**

}

]

})(

**validator**: **this**.*validatePwd*

<Input

ype="lock" style={{color: 'rgba(0,0,0,.25)'}}/>}

|  |  |
| --- | --- |
| **prefix=**{ | <**Icon t** |
| **type="password"** | |

**placeholder="**密码**"**

### ———————————————————————————

/>

)

}

</**Form.Item**>

<Form.Item>

<Button type="primary" htmlType="submit" className="login-form-button">

登陆

</**Button**>

</**Form.Item**>

</**Form**>

</**section**>

</**div**>

)

}

}

*/\**

1. 高阶函数
   1. 一类特别的函数
      1. 接受函数类型的参数
      2. 返回值是函数
   2. 常见
      1. 定时器*: setTimeout()/setInterval()*
      2. *Promise: Promise(() => {}) then(value => {}, reason => {})*
      3. 数组遍历相关的方法*: forEach()/filter()/map()/reduce()/find()/findIndex()*
      4. 函数对象的*bind()*
      5. *Form.create()() / getFieldDecorator()()*
   3. 高阶函数更新动态*,* 更加具有扩展性
2. 高阶组件
   1. 本质就是一个函数
   2. 接收一个组件*(*被包装组件*),* 返回一个新的组件*(*包装组件*),* 包装组件会向被包装组件传入特

定属性

1. 作用*:* 扩展组件的功能
2. 高阶组件也是高阶函数*:* 接收一个组件函数*,* 返回是一个新的组件函数

*\*/*

*/\**

包装*Form* 组件生成一个新的组件*: Form(Login)*

新组件会向*Form* 组件传递一个强大的对象属性*: form*

*\*/*

**const** WrapLogin = Form.*create*()(Login)

### ———————————————————————————

export default connect(

state => ({**user**: state.**user**}),

{*login*}

)(WrapLogin)

.**error-msg** { **visibility**: **hidden**; **position**: **absolute**; **top**: 0;

**left**: 0;

**text-align**: **center**; **height**: 30**px**; **width**: 100%;

**background**: **#f60c1a**; **color**: **#ffffff**;

**font-size**: 16**px**;

**transform**: **translateY**(-30**px**); **transition**: **all** .3**s**;

**&**.**show** {

**visibility**: **visible**; **transform**: **translateY**(0);

}

}

* + 1. **pages/role/role.jsx**

**import** React, {Component} **from 'react' import** {

Card, Button, Table, Modal, message

} from 'antd'

import {connect} from 'react-redux'

**import** {PAGE\_SIZE} **from "../../utils/constants"**

———————————————————————————

**import** {*reqRoles*, *reqAddRole*, *reqUpdateRole*} **from '../../api'**

**import** AddForm **from './add-form' import** AuthForm **from './auth-form'**

**import** {*formateDate*} **from '../../utils/dateUtils' import** {*logout*} **from '../../redux/actions'**

*/\**

角色路由

*\*/*

**class** Role **extends** Component {

**state** = {

**roles**: [], *//* 所有角色的列表

**role**: {}, *//* 选中的*role*

**isShowAdd**: **false**, *//* 是否显示添加界面

**isShowAuth**: **false**, *//* 是否显示设置权限界面

}

constructor (props) {

**super**(props)

**this**.**auth** = React.createRef()

}

*initColumn* = () => {

**this**.**columns** = [

{

**title**: **'**角色名称**'**, **dataIndex**: **'name'**

},

{

**title**: **'**创建时间**'**, **dataIndex**: **'create\_time'**,

render: (create\_time) => *formateDate*(create\_time)

},

{

**title**: **'**授权时间**'**, **dataIndex**: **'auth\_time'**,

**render**: *formateDate*

},

{

**title**: **'**授权人**'**,

### ———————————————————————————

dataIndex: 'auth\_name'

},

]

}

*getRoles* = **async** () => {

**const** result = **await** *reqRoles*()

**if** (result.**status**===0) {

**const** roles = result.**data**

**this**.setState({

roles

})

}

}

*onRow* = (role) => {

**return** {

onClick: event => { *//* 点击行

***console***.log(**'row onClick()'**, role)

*// alert('*点击行*')*

**this**.setState({ role

})

},

}

}

*/\**

添加角色

*\*/*

*addRole* = () => {

*//* 进行表单验证*,* 只能通过了才向下处理

**this**.**form**.validateFields(**async** (error, values) => {

**if** (!error) {

*//* 隐藏确认框

**this**.setState({ **isShowAdd**: **false**

})

*//* 收集输入数据

### ———————————————————————————

**const** {roleName} = values

**this**.**form**.resetFields()

*//* 请求添加

**const** result = **await** *reqAddRole*(roleName)

*//* 根据结果提示*/*更新列表显示

**if** (result.**status**===0) { message.success(**'**添加角色成功**'**)

*// this.getRoles()*

*//* 新产生的角色

**const** role = result.**data**

*//* 更新*roles* 状态

*/\*const roles = this.state.roles roles.push(role)*

*this.setState({ roles*

*})\*/*

*//* 更新*roles* 状态*:* 基于原本状态数据更新

**this**.setState(state => ({

**roles**: [...state.**roles**, role]

}))

} **else** {

message.success(**'**添加角色失败**'**)

}

}

})

}

*/\**

更新角色

*\*/*

*updateRole* = **async** () => {

*//* 隐藏确认框

**this**.setState({ **isShowAuth**: **false**

})

### ———————————————————————————

**const** role = **this**.**state**.**role**

*//* 得到最新的 *menus*

**const** menus = **this**.**auth**.**current**.*getMenus*() role.**menus** = menus

role.**auth\_time** = Date.now() role.**auth\_name** = **this**.**props**.user.**username**

*//* 请求更新

**const** result = **await** *reqUpdateRole*(role)

**if** (result.**status**===0) {

*// this.getRoles()*

*//* 如果当前更新的是自己角色的权限*,* 强制退出

**if** (role.**\_id** === **this**.**props**.user.role\_id) {

**this**.**props**.logout()

message.success(**'**当前用户角色权限成功**'**)

} **else** {

message.success(**'**设置角色权限成功**'**)

**this**.setState({

**roles**: [...**this**.**state**.**roles**]

})

}

}

}

componentWillMount () {

**this**.*initColumn*()

}

componentDidMount () {

**this**.*getRoles*()

}

render() {

**const** {**roles**, role, **isShowAdd**, **isShowAuth**} = **this**.**state**

**const** title = (

<**span**>

<**Button type='primary' onClick=**{() => **this**.setState({**isShowAdd**: **true**})}>创建角色</**Button**> **&nbsp;&nbsp;**

### ———————————————————————————

<**Button type='primary' disabled=**{!role.**\_id**} **onClick=**{() =>

**this**.setState({**isShowAuth**: **true**})}>设置角色权限</**Button**>

</**span**>

)

return (

<**Card title=**{title}>

<Table

**rowKey='\_id'**

{**roles**}

{**this**.**columns**}

{{**defaultPageSize**: PAGE\_SIZE}}

**rowSelection=**{{

**type**: **'radio'**, **selectedRowKeys**: [role.**\_id**],

onSelect: (role) => { *//* 选择某个*radio* 时回调

**this**.setState({ role

})

}

}}

**onRow=**{**this**.*onRow*}

<**Modal**

**title="**添加角色**"**

{**isShowAdd**}

{**this**.*addRole*}

**onCancel=**{() => {

**this**.setState({**isShowAdd**: **false**}) **this**.**form**.resetFields()

}}

**onOk=**

**visible=**

**pagination=**

**columns=**

**dataSource=**

>

/>

**bordered**

<AddForm

**setForm=**{(form) => **this**.**form** = form}

/>

</**Modal**>

<Modal

**title="**设置角色权限**"**

**visible=**{**isShowAuth**}

**onOk=**{**this**.*updateRole*} **onCancel=**{() => {

**this**.setState({**isShowAuth**: **false**})

}}

>

<**AuthForm ref=**{**this**.**auth**} **role=**{role}/>

</**Modal**>

</**Card**>

)

}

}

**export default connect**(

state => ({**user**: state.**user**}),

{*logout*}

)(Role)

# 可视化图表

### 常用数据可视化图表库

###### echarts

* 1. <https://echarts.baidu.com/>
  2. 百度开源, 如果要在 react 项目中使用, 需要下载 echarts-for-react

1. G2
   1. <https://antv.alipay.com/zh-cn/g2/3.x/index.html>
   2. 阿里开源
2. bizcharts
   1. <https://bizcharts.net/products/bizCharts>
   2. 基于 react 包装 G2 的开源库
   3. 需要额外下载 @antv/data-set
3. d3
   1. <https://d3js.org.cn/>
   2. 国外的免费可视化图表库
      1. 下载依赖

**yarn add echarts echarts-for-react**

**yarn add bizcharts @antv/data-set**

* + 1. **bar.jsx:** 柱状图

**import** React, {Component} **from 'react' import** {Card, Button} **from 'antd'**

**import** ReactEcharts **from 'echarts-for-react'**

*/\**

后台管理的柱状图路由组件

*\*/*

**export default class** Bar **extends** Component {

**state** = {

**sales**: [5, 20, 36, 10, 10, 20],

**inventorys**: [15, 30, 46, 20, 20, 40]

}

*getOption* = () => {

**const** {**sales**, **inventorys**} = **this**.**state return** {

**title**: {

**text**: **'ECharts** 入门示例**'**

},

**tooltip**: {},

**legend**: {

**data**:[**'**销量**'**, **'**库存**'**]

},

**xAxis**: {

**data**: [**"**衬衫**"**,**"**羊毛衫**"**,**"**雪纺衫**"**,**"**裤子**"**,**"**高跟鞋**"**,**"**袜子**"**]

},

**yAxis**: {},

**series**: [{

**name**: **'**销量**'**,

**type**: **'bar'**,

**data**:**sales**

}, {

**name**: **'**库存**'**,

**type**: **'bar'**, **data**: **inventorys**

}]

}

}

*update* = () => {

**const** sales = **this**.**state**.**sales**.map(sale => sale + 1)

**const** inventorys = **this**.**state**.**inventorys**.map(inventory => inventory -1)

**this**.setState({ sales, inventorys

})

}

render() {

**return** (

<**div**>

<**Card**>

<**Button type='primary' onClick=**{**this**.*update*}>更新</**Button**>

</**Card**>

<**Card title='**柱状图一**'**>

<**ReactEcharts option=**{**this**.*getOption*()} **style=**{{**height**: 300}}/>

</**Card**>

</**div**>

)

}

}

* + 1. **line.jsx:** 折线图

**import** React, {Component} **from 'react' import** {Card, Button} **from 'antd'**

**import** ReactEcharts **from 'echarts-for-react'**

*/\**

后台管理的折线图路由组件

### ———————————————————————————

*\*/*

**export default class** Line **extends** Component {

**state** = {

**sales**: [5, 20, 36, 10, 10, 20],

**inventorys**: [15, 30, 46, 20, 20, 40]

}

*getOption* = () => {

**const** {**sales**, **inventorys**} = **this**.**state return** {

**title**: {

**text**: **'ECharts** 入门示例**'**

},

**tooltip**: {},

**legend**: {

**data**:[**'**销量**'**, **'**库存**'**]

},

**xAxis**: {

**data**: [**"**衬衫**"**,**"**羊毛衫**"**,**"**雪纺衫**"**,**"**裤子**"**,**"**高跟鞋**"**,**"**袜子**"**]

},

**yAxis**: {},

**series**: [{

**name**: **'**销量**'**,

**type**: **'line'**, **data**:**sales**

}, {

**name**: **'**库存**'**,

**type**: **'line'**, **data**: **inventorys**

}]

}

}

*getOption2* = () => {

**return** {

**xAxis**: {

**type**: **'category'**, **boundaryGap**: **false**,

**data**: [**'Mon'**, **'Tue'**, **'Wed'**, **'Thu'**, **'Fri'**, **'Sat'**, **'Sun'**]

},

**yAxis**: {

**type**: **'value'**

},

**series**: [{

**data**: [820, 932, 901, 934, 1290, 1330, 1320],

**type**: **'line'**, **areaStyle**: {}

}]

};

}

*update* = () => {

**const** sales = **this**.**state**.**sales**.map(sale => sale + 1)

**const** inventorys = **this**.**state**.**inventorys**.map(inventory => inventory -1)

**this**.setState({ sales, inventorys

})

}

render() {

**return** (

<**div**>

<**Card**>

<**Button type='primary' onClick=**{**this**.*update*}>更新</**Button**>

</**Card**>

<**Card title='**折线图一**'**>

<**ReactEcharts option=**{**this**.*getOption*()} **style=**{{**height**: 300}}/>

</**Card**>

<**Card title='**折线图二**'**>

<**ReactEcharts option=**{**this**.*getOption2*()} **style=**{{**height**: 300}}/>

</**Card**>

</**div**>

)

}

}

* + 1. **pie.jsx:** 饼状图

**import** React, {Component} **from 'react' import** {Card} **from 'antd'**

**import** ReactEcharts **from 'echarts-for-react'**

*/\**

后台管理的饼图路由组件

*\*/*

**export default class** Pie **extends** Component {

*getOption* = () => {

**return** {

**title** : {

**text**: **'**某站点用户访问来源**'**, **subtext**: **'**纯属虚构**'**,

**x**:**'center'**

},

**tooltip** : {

**trigger**: **'item'**,

**formatter**: **"{a} <br/>{b} : {c} ({d}%)"**

},

**legend**: {

**orient**: **'vertical'**, **left**: **'left'**,

**data**: [**'**直接访问**'**,**'**邮件营销**'**,**'**联盟广告**'**,**'**视频广告**'**,**'**搜索引擎**'**]

},

**series** : [

{

**name**: **'**访问来源**'**,

**type**: **'pie'**,

**radius** : **'55%'**,

**center**: [**'50%'**, **'60%'**],

**data**:[

{**value**:335, **name**:**'**直接访问**'**},

{**value**:310, **name**:**'**邮件营销**'**},

{**value**:234, **name**:**'**联盟广告**'**},

{**value**:135, **name**:**'**视频广告**'**},

{**value**:1548, **name**:**'**搜索引擎**'**}

],

**itemStyle**: {

**emphasis**: {

**shadowBlur**: 10,

**shadowOffsetX**: 0,

shadowColor: 'rgba(0, 0, 0, 0.5)'

}

}

}

]

};

}

*getOption2* = () => {

**return** {

**backgroundColor**: **'#2c343c'**,

**title**: {

**text**: **'Customized Pie'**, **left**: **'center'**,

**top**: 20,

**textStyle**: {

**color**: **'#ccc'**

}

},

**tooltip** : {

**trigger**: **'item'**,

**formatter**: **"{a} <br/>{b} : {c} ({d}%)"**

},

**visualMap**: {

**show**: **false**, **min**: 80,

**max**: 600,

**inRange**: {

**colorLightness**: [0, 1]

}

},

**series** : [

{

**name**:**'**访问来源**'**,

type:'pie',

### ———————————————————————————

**radius** : **'55%'**,

**center**: [**'50%'**, **'50%'**],

**data**:[

{**value**:335, **name**:**'**直接访问**'**},

{**value**:310, **name**:**'**邮件营销**'**},

{**value**:274, **name**:**'**联盟广告**'**},

{**value**:235, **name**:**'**视频广告**'**},

{**value**:400, **name**:**'**搜索引擎**'**}

].sort(**function** (a, b) { **return** a.**value** - b.**value**; }),

**roseType**: **'radius'**, **label**: {

**normal**: {

**textStyle**: {

**color**: **'rgba(255, 255, 255, 0.3)'**

}

}

},

**labelLine**: {

**normal**: {

**lineStyle**: {

**color**: **'rgba(255, 255, 255, 0.3)'**

},

**smooth**: 0.2,

**length**: 10,

**length2**: 20

}

},

**itemStyle**: {

**normal**: {

**color**: **'#c23531'**, **shadowBlur**: 200,

**shadowColor**: **'rgba(0, 0, 0, 0.5)'**

}

},

**animationType**: **'scale'**, **animationEasing**: **'elasticOut'**, animationDelay: **function** (idx) {

**return *Math***.random() \* 200;

}

}

]

### ———————————————————————————

**import** React **from "react" import** {

Chart, Geom, Axis, Tooltip, Legend,

} from "bizcharts"

**import** DataSet **from "@antv/data-set"**

**export default class** Line **extends** React.Component { render() {

**const** data = [

{

**month**: **"Jan"**,

**a**: 7.0,

**b**: 3.9,

**c**: 5.9

};

}

render() {

return (

<**div**>

<**Card title='**饼图一**'**>

<**ReactEcharts option=**{**this**.*getOption*()} **style=**{{**height**: 300}}/>

</**Card**>

<**Card title='**饼图二**'**>

<**ReactEcharts option=**{**this**.*getOption2*()} **style=**{{**height**: 300}}/>

</**Card**>

</**div**>

)

}

}

* + 1. 首页图表

###### home/line.jsx

———————————————————————————

{

**month**: **"Feb"**,

**a**: 6.9,

**b**: 4.2,

**c**: 1.9

},

{

**month**: **"Mar"**,

**a**: 9.5,

**b**: 5.7,

**c**: 3.9

},

{

**month**: **"Apr"**,

**a**: 14.5,

**b**: 8.5,

**c**: 5.5

},

{

**month**: **"May"**,

**a**: 18.4,

**b**: 11.9,

**c**: 8.9

},

{

**month**: **"Jun"**,

**a**: 21.5,

**b**: 15.2,

**c**: 10.0

},

{

**month**: **"Jul"**,

**a**: 25.2,

**b**: 17.0,

**c**: 12.9

},

{

**month**: **"Aug"**,

**a**: 26.5,

**b**: 16.6,

**c**: 15.9

### ———————————————————————————

{

**month**: **"Sep"**,

**a**: 23.3,

**b**: 14.2,

**c**: 20.7

},

{

**month**: **"Oct"**,

**a**: 18.3,

**b**: 10.3,

**c**: 25.9

},

{

**month**: **"Nov"**,

**a**: 13.9,

**b**: 6.6,

**c**: 30.9

},

{

**month**: **"Dec"**,

**a**: 9.6,

**b**: 4.8,

**c**: 35.9

}

]

**const** ds = **new** DataSet()

**const** dv = ds.createView().source(data) dv.transform({

**type**: **"fold"**,

**fields**: [**"a"**, **"b"**, **"c"**],

*//* 展开字段集

**key**: **"city"**,

*// key* 字段

**value**: **"temperature"** *// value* 字段

})

**const** cols = {

**month**: {

**range**: [0, 1]

}

}

return (

<**div style=**{{**float**: **'right'**, **width**: 750, **height**: 300}}>

### ———————————————————————————

<**Chart height=**{250} **data=**{dv} **scale=**{cols} **forceFit**>

<**Legend**/>

<**Axis name="month"**/>

<**Axis**

**name="temperature" label=**{{

formatter: val => **`**${val}万个**`**

}}

<**Tooltip**

**crosshairs=**{{

**type**: **"y"**

}}

/>

<**Geom**

**position="month\*temperature" size=**{2}

**color=**{**"city"**} **shape=**{**"smooth"**}

/>

<**Geom**

**position="month\*temperature" size=**{4}

**shape=**{**"circle"**} **color=**{**"city"**} **style=**{{

**stroke**: **"#fff"**, **lineWidth**: 1

}}

/>

</**Chart**>

</**div**>

)

}

}

**type="point"**

**type="line"**

/>

###### home/bar.jsx

———————————————————————————

**import** React **from "react"**

**import** { Chart, Geom, Axis, Tooltip,

} from "bizcharts"

**export default class** Bar **extends** React.Component {

render() {

**const** data = [

{

**year**: **"1** 月**"**,

**sales**: 38

},

{

**year**: **"2** 月**"**,

**sales**: 52

},

{

**year**: **"3** 月**"**,

**sales**: 61

},

{

**year**: **"4** 月**"**,

**sales**: 145

},

{

**year**: **"5** 月**"**,

**sales**: 48

},

{

**year**: **"6** 月**"**,

**sales**: 38

},

{

**year**: **"7** 月**"**,

**sales**: 28

},

{

**year**: **"8** 月**"**,

**sales**: 38

### ———————————————————————————

},

{

**year**: **"59** 月**"**,

**sales**: 68

},

{

**year**: **"10** 月**"**,

**sales**: 38

},

{

**year**: **"11** 月**"**,

**sales**: 58

},

{

**year**: **"12** 月**"**,

**sales**: 38

}

]

**const** cols = {

**sales**: {

**tickInterval**: 20

}

}

**return** (

<**div style=**{{**width**: **'100%'**, **marginLeft**: -30}}>

<**Chart height=**{338} **data=**{data} **scale=**{cols} **forceFit**>

<**Axis name="year"**/>

<**Axis name="sales"**/>

<**Tooltip**

**crosshairs=**{{

**type**: **"y"**

}}

/>

<**Geom type="interval" position="year\*sales"**/>

</**Chart**>

</**div**>

)

}

}

###### home/home.less

———————————————————————————

.**home** {

**padding**: 24**px**; **background**: **#fff**; **min-height**: 850**px**;

.**home-card** {

**float**: **left**;

}

.**home-content** { **position**: **absolute**; **top**: 420**px**;

**width**: 76%;

**border**: 1**px solid #e8e8e8**;

.**home-menu** {

**font-size**: 20**px**;

**span** {

**cursor**: **pointer**;

}

.**home-menu-active** {

**border-bottom**: 2**px solid #1DA57A**; **color**: **#1DA57A**;

**padding**: 0 0 16**px** 0;

}

.**home-menu-visited** {

**margin-right**: 40**px**;

}

}

.**home-table-left** { **float**: **left**; **width**: 60%;

}

.**home-table-right** { **float**: **right**; **width**: 330**px**;

}

}

}

1. Home/home.jsx

**import** React, {Component} **from 'react' import** {

Icon,

———————————————————————————

Card,

Statistic, DatePicker, Timeline

} from 'antd'

**import** moment **from 'moment'**

import Line from './line' import Bar from './bar'

**import './home.less'**

**const** dateFormat = **'YYYY/MM/DD'**

**const** {RangePicker} = DatePicker

**export default class** Home **extends** Component {

**state** = {

**isVisited**: **true**

}

*handleChange* = (isVisited) => {

**return** () => **this**.setState({isVisited})

}

render() {

**const** {**isVisited**} = **this**.**state**

**return** (

<**div className='home'**>

<**Card**

**className="home-card"**

<**Icon style=**{{**color**: **'rgba(0,0,0,.45)'**}} **type="question-circle"**/>}

{**width**: 250}}

**style=**{

**extra=**{

**title="**商品总量**"**

{{**color**: **'rgba(0,0,0,.45)'**}}

>

headStyle=

<**Statistic value=**{1128163}

**suffix="**个**"**

{{**fontWeight**: **'bolder'**}}

/>

style=

<Statistic

### ———————————————————————————

**value=**{15}

**valueStyle=**{{**fontSize**: 15}}

|  |  |  |
| --- | --- | --- |
| **prefix=**{ | **'**周同比**'**} | |
| **suffix=**{ | <**div**> | %<**Icon style=** |

{{**color**: **'red'**, **marginLeft**: 10}}

**type="arrow-down"**/></**div**>}

/>

<Statistic

{10}

value=

{{**fontSize**: 15}}

**'**日同比**'**}

<**div**>%<**Icon style=**{{**color**: **'#3f8600'**, **marginLeft**: 10}}

**type="arrow-up"**/></**div**>}

</**Card**>

<**Line**/>

<**Card**

<**div className="home-menu"**>

<**span className=**{**isVisited** ? **"home-menu-active home-menu-visited"** :

**'home-menu-visited'**}

**onClick=**{**this**.*handleChange*(**true**)}>访问量</**span**>

<**span className=**{**isVisited** ? **""** : **'home-menu-active'**} **onClick=**{**this**.*handleChange*(**false**)}>销售量</**span**>

</**div**>}

<**RangePicker**

{[moment(**'2019/01/01'**, dateFormat), moment(**'2019/06/01'**,

dateFormat)]}

**format=**{dateFormat}

}

<**Card**

**className="home-table-left" title=**{**isVisited** ? **'**访问趋势**'** : **'**销售趋势**'**}

>

/>

**defaultValue=**

**extra=**{

**title=**{

**className="home-content"**

**suffix=**{

**prefix=**{

/>

valueStyle=

{{**padding**: 0, **height**: 275}}

**extra=**{<**Icon type="reload"**/>}

>

bodyStyle=

<**Bar**/>

</**Card**>

<Card title='任务' extra={<Icon type="reload"/>}

### ———————————————————————————

className="home-table-right">

<Timeline>

<**Timeline.Item color="green"**>新版本迭代会</**Timeline.Item**>

<**Timeline.Item color="green"**>完成网站设计初版</**Timeline.Item**>

<Timeline.Item color="red">

<**p**>联调接口</**p**>

<**p**>功能验收</**p**>

</Timeline.Item>

<Timeline.Item>

<**p**>登录功能设计</**p**>

<**p**>权限验证</**p**>

<**p**>页面排版</**p**>

</Timeline.Item>

</**Timeline**>

</**Card**>

</**Card**>

</**div**>

)

}

}

* 1. 前台 **404** 界面
     1. **not-found/not-found.jsx**

**import** React, {Component} **from 'react' import** {Button, Row, Col} **from 'antd' import** {**connect**} **from 'react-redux'**

**import** {*setHeadTitle*} **from '../../redux/actions' import './not-found.less'**

*/\**

前台*404* 页面

*\*/*

**class** NotFound **extends** Component {

### ———————————————————————————

*goHome* = () => { **this**.**props**.setHeadTitle(**'**首页**'**) **this**.**props**.history.replace(**'/home'**)

}

render() {

return (

<Row className='not-found'>

<**Col span=**{12} **className='left'**></**Col**>

<**Col span=**{12} **className='right'**>

<**h1**>404</**h1**>

<**h2**>抱歉，你访问的页面不存在</**h2**>

<**div**>

<**Button type='primary' onClick=**{**this**.*goHome*}>

回到首页

</**Button**>

</**div**>

</**Col**>

</**Row**>

)

}

}

export default connect( null,

{*setHeadTitle*}

)(NotFound)

* + 1. **not-found/not-found.less**

.not-found{

**background-color**: **#f0f2f5**; **height**: 100%;

.**left** {

**height**: 100%;

background: url('./images/404.png') no-repeat center;

}

.**right** {

**padding-left**: 50**px**;

**margin-top**: 150**px**;

**h1** {

**font-size**: 35**px**;

}

**h2** {

**margin-bottom**: 20**px**;

**font-size**: 20**px**;

}

}

}

* + 1. **admin/admin.jsx**

<Redirect from='/' to='/home' exact/>

<**Route path='/home' component=**{Home}/>

<**Route path='/category' component=**{Category}/>

<**Route path='/product' component=**{Product}/>

<**Route path='/role' component=**{Role}/>

<**Route path='/user' component=**{User}/>

<Route path='/charts/bar' component={Bar}/>

<Route path='/charts/line' component={Line}/>

<Route path='/charts/pie' component={Pie}/>

<**Route component=**{NotFound}/>

* 1. 打包项目并运行
     1. 打包项目

**yarn run build**

**2.23.1.** 运行打包项目

##### 与服务器端项目独立运行

###### 问题: 存在 ajax 请求跨域问题

解决: 由服务器端工程师配置代理服务器(前端工程师不用亲自操作)

1. 合并到服务端项目一起运行

不再有 ajax 请求跨域问题

1. 使用 **BrowserRouter** 的问题

###### 问题: 刷新某个路由路径时, 会出现 404 的错误

* 1. 原因: 项目根路径后的 path 路径会被当作后台路由路径, 去请求对应的后台路由, 但没有
  2. 解决: 使用自定义中间件去读取返回 index 页面展现

**const** fs = *require*(**'fs'**)

// 必须在路由器中间之后声明使用

app.use((req, res) => {

fs.readFile( dirname + **'/public/index.html'**, (err, data)=>{

**if**(err){

***console***.log(err) res.send(**'**后台错误**'**)

} **else** { res.writeHead(200, {

'Content-Type': 'text/html; charset=utf-8',

});

res.**end**(data)

}

})

})

* 1. 注意: 前端路由的路径不要与后台路由路径相同(并且请求方式也相同)