

七、企业管理后台系统开发

1、主页面搭建

1.1 Ant Design基本使用

整个管理后台的搭建，我们借助了Ant Design来完成布局的。

官方地址：

<https://ant.design/index-cn>

对应包的安装

```
npm install antd --save
```

安装对应的包以后，就可以在项目中使用了。

在这里我们在项目的App.js文件中使用一下。

```
import {Button} from 'antd'  
import 'antd/dist/antd.css'  
<Button>Hello world</Button>
```

导入Button组件，同时引入对应的css样式，这样就可以使用对应的Button这个组件了。

但是这里有一个问题就是，我们在引用样式的时候，是全部引用了，由于样式文件比较大，这样加载起来就比较慢了。所以，这里我们按需引入，也就是说，如果我只用了Button这个组件，那么我们只需要引入Button组件所需要的样式就可以了。这样可以提升性能。

这里需要安装一个插件：babel-plugin-import

[babel-plugin-import](#) 是一个用于按需加载组件代码和样式的 babel 插件

```
npm install babel-plugin-import --save-dev
```

该插件安装完成后，需要进行配置，也就是在webpack中进行配置，但是我们创建完项目后，默认的webpack会隐藏，那么怎样把该配置内容展示出来呢？

这时，打开package.json文件，在对应的"scripts"这一项中有一个命令是eject,运行该命令就可以看到项目中的配置文件。（在当前项目所在的目录下，启动命令窗口，执行如下命令）

```
npm run eject
```

下面需要对babel-plugin-import这个插件进行配置，实现样式的按需加载。

找到package.json文件，找到该文件中的，如下项目

```
"babel": {  
  "presets": [  
    "react-app"  
  ]  
}
```

在该项目中添加如下的配置：

```
"plugins":  
[  
  [  
    "import",  
    {  
      "libraryName": "antd",  
      "libraryDirectory": "es",  
      "style": "css"  
    }  
  ]  
]  
]完整的配置如下：
```

```
"babel": {  
  "presets": [  
    "react-app"  
  ],  
  "plugins":  
  [  
    [  
      "import",  
      {  
        "libraryName": "antd",  
        "libraryDirectory": "es",  
        "style": "css"  
      }  
    ]  
  ]  
}
```

```
    }  
  ]  
]  
  
}
```

这时完成了样式的按需加载配置。

注意：然后，回到App.js文件中，把antd.css文件的引入去掉。那么这时，重新启动服务器，刷新浏览器发现样式起作用了，这样就完成了样式的按需加载的配置。

1.2 整体结构创建

接下来创建项目的目录结构，整个项目的后台主页面分为了上，下，左，中。而且每个页面中展示这部分内容都是

相同的，所以可以把这几块区域都封装成公共的组件，所以在项目中创建一个文件夹components，把公共的组件都放在该文件夹下（每个组件都对应的是一个目录，而目录下创建对应的js文件与css文件）。

然后在项目的src目录下创建一个admin.js文件，作为后台首页中的整体布局页面。

从大的方面来讲，主页面分为了左右两部分，所以这里可以使用栅格布局的方式。

栅格布局：<https://ant.design/components/grid-cn/>

在admin.js文件中，通过栅格布局，将其分为左右两部分内容。

```
import React, { Component } from 'react';  
import { Row, Col } from 'antd'  
class Admin extends Component {  
  render() {  
    return (  
      <Row>  
        <Col span={3}>  
          Left  
        </Col>  
        <Col span={21}>  
          Right  
        </Col>  
      </Row>  
    )  
  }  
}  
export default Admin
```

整个布局中，所有列相加为24。

下面在对右侧的区域再次进行拆分。分为了顶部区域，中间区域以及底部区域。对应的代码如下：

```
import React, { Component } from 'react';  
import { Row, Col } from 'antd'  
import Header from './components/Header'  
import Footer from './components/Footer'  
class Admin extends Component {  
  render() {  
    return (  
      <Row>  
        <Col span={3}>  
          Left  
        </Col>  
        <Col span={21}>  
          <Header>  
  
          </Header>  
          <Row>  
            中间内容区域  
          </Row>  
          <Footer></Footer>  
        </Col>  
      </Row>  
    )  
  }  
}  
export default Admin
```

对应的Header组件代码如下：

```
import React, { Component } from 'react'
class Header extends Component {
  render() {
    return (
      <div>
        这是头部区域
      </div>
    )
  }
}
export default Header
```

对应的Footer组件的代码如下：

```
import React, { Component } from 'react'
class Footer extends Component {
  render() {
    return (
      <div>
        这是底部区域
      </div>
    )
  }
}
export default Footer
```

同时NavLeft组件的内容如下：

```
import React, { Component } from 'react'
class NavLeft extends Component {
  render() {
    return (
      <div>
        这是左侧菜单区域
      </div>
    )
  }
}
export default NavLeft
```

对应的在admin.js文件中使用NavLeft组件。

```
import React, { Component } from 'react';
import { Row, Col } from 'antd'
import Header from './components/Header'
import Footer from './components/Footer'
import NavLeft from './components/NavLeft' //导入NavLeft组件
class Admin extends Component {
  render() {
    return (
      <Row>
        <Col span={3}>
          <NavLeft/> //展示左侧的内容
        </Col>
        <Col span={21}>
          <Header>
            </Header>
          <Row>
            中间内容区域
          </Row>
          <Footer></Footer>
        </Col>
      </Row>
    )
  }
}
export default Admin
```

现在基本的结构已经创建完成了，下面可以添加一下基本的样式。

在src目录下，创建一个style目录，创建一个common.css样式文件。

```
.container{
  display: flex;
}
.nav-left{
  width: 15%;
  min-width: 180px;
  height: calc(100vh);
  background-color: blue;
}
.main{
  flex: 1;
```

```

    height: calc(100vh);
  }
  .content{
    position: relative;
    padding:20px;
  }
}

```

对应的在admin.js文件中导入该样式文件，同时为Row, Col添加具体的样式

```

import React, { Component } from 'react';
import { Row, Col } from 'antd'
import Header from './components/Header'
import Footer from './components/Footer'
import NavLeft from './components/NavLeft'
import './style/common.css'
class Admin extends Component{
  render(){
    return (
      <Row className='container'>
        <Col span={3} className="nav-left">
          <NavLeft/>
        </Col>
        <Col span={21} className="main">
          <Header>

          </Header>
          <Row className="content">
            中间内容区域
          </Row>
          <Footer></Footer>
        </Col>
      </Row>
    )
  }
}
export default Admin

```

1.3 菜单处理

1.3.1 菜单布局

Menu菜单地址: <https://ant.design/components/menu-cn/>

具体实现的代码如下: (NavLeft/index.js)

```

import React, { Component } from 'react'
import { Menu, Icon } from 'antd';
import './index.css'
const { SubMenu } = Menu;
class NavLeft extends Component {
  render() {
    return (
      <div>
        { /* Logo图片的布局 */ }
        <div className="logo">
          
          <h1>后台管理系统</h1>
        </div>
        { /* 左侧菜单实现布局 */ }
        <Menu theme="dark">
          <SubMenu
            key="sub1"
            title={
              <span>
                <Icon type="mail" />
                <span>Navigation One</span>
              </span>
            >
            <Menu.Item key="1">Option 1</Menu.Item>
            <Menu.Item key="2">Option 2</Menu.Item>
            <Menu.Item key="3">Option 3</Menu.Item>
            <Menu.Item key="4">Option 4</Menu.Item>
          </SubMenu>
        </Menu>
      </div>
    )
  }
}
export default NavLeft

```

初步的样式设计 (NavLeft/index.css) :

```

.logo{
  line-height:90px;
  padding-left: 20px;
}
img{
  height: 100px;
}
h1{
  color:#ffff;
  font-size: 20px;
  margin:0 0 0 10px;
  display:inline-block;
  vertical-align: middle;
}

```

1.3.2 动态加载菜单项

将所有的菜单定义了src/config/menuConfig.js文件中，是一个对象数组。

如下所示：

```

const menuList = [
  {
    title: '首页',
    key: '/home'
  },
  {
    title: '常用组件',
    key: '/admin/componet',
    children: [
      {
        title: '按钮',
        key: '/admin/componet/buttons',
      },
      {
        title: '弹框',
        key: '/admin/componet/modals',
      },
      {
        title: 'Loading',
        key: '/admin/componet/loadings',
      },
      {
        title: '通知提醒',
        key: '/admin/componet/notification',
      },
      {
        title: 'Tab页签',
        key: '/admin/componet/tabs',
      }
    ]
  },
  {
    title: '表单',
    key: '/form',
    children: [
      {
        title: '登录',
        key: '/form/login',
      },
      {
        title: '注册',
        key: '/form/reg',
      }
    ]
  },
  {
    title: '表格',
    key: '/table',
    children: [
      {
        title: '基础表格',
        key: '/table/basic',
      },
      {
        title: '高级表格',
        key: '/table/high',
      }
    ]
  },
  {
    title: '订单管理',
    key: '/order',
    btnList: [
      {

```

```

        title: '订单详情',
        key: 'detail'
      },
      {
        title: '结束订单',
        key: 'finish'
      }
    ]
  },
  {
    title: '用户管理',
    key: '/user'
  },
  {
    title: '图标',
    key: '/charts',
    children: [
      {
        title: '柱形图',
        key: '/charts/bar'
      },
      {
        title: '饼图',
        key: '/charts/pie'
      },
      {
        title: '折线图',
        key: '/charts/line'
      }
    ]
  }
];
export default menuList;

```

下面要处理的是，在NavLeft/index.js文件中加载菜单项。

首先先导入对应的文件

```
import MenuConfig from '.././config/menuConfig'
```

接下来创建一个state,存储加载出来的菜单数据。

```

class NavLeft extends Component {
  constructor(props) {
    super(props)
    this.state={
      menuTree: []
    }
  }
}

```

同时在NavLeft这个组件中，定使用componentDidMount这个生命周期函数，加载对应的菜单项。

把加载的菜单项赋值给了对应的state.

```

componentDidMount() {
  const menuTree=this.loadMenu(MenuConfig)
  this.setState({menuTree})
}

```

loadMenu这个方法，是通过递归的方式，加载对应的菜单项。

```

//加载菜单,通过递归的方式
loadMenu=(data)=>{
  return data.map((item)=>{
    if(item.children){
      return(
        <SubMenu title={item.title} key={item.key}>
          {this.loadMenu(item.children)}
        </SubMenu>
      )
    }
    return <Menu.Item title={item.title} key={item.key}>{item.title}</Menu.Item>
  })
}

```

最后在render方法中展示出对应的菜单项。

```

<Menu theme="dark">
  {this.state.menuTree}
</Menu>

```

具体实现的代码如下：

```
import React, { Component } from 'react'
import { Menu, Icon } from 'antd';
import './index.css'
import MenuConfig from '.././config/menuConfig'
const { SubMenu } = Menu;

class NavLeft extends Component {
  constructor(props) {
    super(props)
    this.state={
      menuTree: []
    }
  }
  componentDidMount() {
    const menuTree=this.loadMenu(MenuConfig)
    this.setState({menuTree})
  }
  //加载菜单,通过递归的方式
  loadMenu=(data)=>{
    return data.map((item)=>{
      if(item.children){
        return(
          <SubMenu title={item.title} key={item.key}>
            {this.loadMenu(item.children)}
          </SubMenu>
        )
      }
      return <Menu.Item title={item.title} key={item.key}>{item.title}</Menu.Item>
    })
  }
  render() {
    return (
      <div>
        { /* Logo图片的布局 */ }
        <div className="logo">
          
          <h1>后台管理系统</h1>
        </div>
        { /* 左侧菜单实现布局 */ }
        <Menu theme="dark">
          {this.state.menuTree}
        </Menu>

        { /* <Menu theme="dark">
          <SubMenu
            key="sub1"
            title={
              <span>
                <Icon type="mail" />
                <span>Navigation One</span>
              </span>
            >
          <Menu.Item key="1">Option 1</Menu.Item>
          <Menu.Item key="2">Option 2</Menu.Item>
          <Menu.Item key="3">Option 3</Menu.Item>
          <Menu.Item key="4">Option 4</Menu.Item>
        </SubMenu>
        </Menu> */ }

      </div>
    )
  }
}
export default NavLeft
```

1.4 顶部区域的实现

顶部区域布局

```
import React, { Component } from 'react'
import { Row, Col } from 'antd'
import './index.css'
class Header extends Component {
  constructor(props) {
    super(props)
    this.state={
      userName: '张三'
    }
  }
}
```

```

render() {
  return (
    <div>
      { /* 第一行内容 */ }
      <Row className="header-top">
        <Col span={24}>
          <span>欢迎: {this.state.userName} 登录系统</span>
          <a href="#">退出</a>
        </Col>
      </Row>
      { /* 第二行内容 */ }
      <Row className='breadcrumb'>
        <Col span={4} className='breadcrumb-title'>
          首页
        </Col>
        <Col span={20} className='breadcrumb-date'>
          <span>2119年-11月-1日</span>
        </Col>
      </Row>
    </div>
  )
}
export default Header

```

下面创建对应的样式

在Header/index.css

```

.header-top{
  height: 60px;
  line-height: 60px;
  padding: 0 30px;
  text-align: right;
}
.header-top a{
  margin-left: 50px;
  font-size: 14px;
}
.breadcrumb{
  height: 50px;
  line-height: 50px;
  padding: 0 30px;
  border-top: 1px solid blue;
}
.breadcrumb-title{
  text-align: center;
  font-size: 18px;
  font-weight: bold;
}
.breadcrumb-date{
  text-align: right;
  font-size: 16px;
  font-weight: bold;
}

```

1.5 底部区域的实现

底部内容非常简单，就是展示版权信息。

```

import React, { Component } from 'react'
import './index.css'
class Footer extends Component {
  render() {
    return (
      <div className='footer'>
        江苏传智播客教育科技股份有限公司 Copyright 2006-2019, All Rights Reserved 苏ICP备16007882号-12 营业
        执照 增值电信业务经营许可证
      </div>
    )
  }
}
export default Footer

```

对应的样式如下 (Footer/index.css) :


```
.footer{
  height: 100px;
  padding:40px 0;
  text-align: center;
  font-size: 14px;
  color: darkgray;
}
```

1.6 内容区域的实现

内容区域不是一个公共的组件，因为当单击不同的菜单项的时候，对应的内容都会展示在该区域。

所以这里，我们不能将该内容放到components目录下。在src目录下创建一个home目录，来存放对应的内容。

对应的内容如下：

```
import React, { Component } from 'react'
import './index.css'
export default class Home extends Component{
  render(){
    return (
      <div className="home-style">
        欢迎访问系统的后台管理
      </div>
    )
  }
}
```

对应的样式如下：

```
.home-style{
  height: calc(70%);
  display: flex;
  align-items: center;
  justify-content: center;
  font-size:20px;
}
```

在回到src/admin.js文件，去导入内容区域。

```
import Home from './home'
<Row className="content">
  <Home/>
</Row>
```

2、路由设置

2.1 路由基本设置

这里需要在项目中安装路由

```
npm add react-router-dom
```

然后在项目的src目录下创建router.js文件，在该文件中进行路由的配置

```
import React, { Component } from 'react'
import {BrowserRouter,Route} from 'react-router-dom'
import App from './App'
import Admin from './admin'
import Login from './pagecomponent/login'
class IRouter extends Component{
  render(){
    return (
      <BrowserRouter>
        <App>
          <Route path="/login" component={Login} />
          <Route path="/admin" component={Admin} />
        </App>
      </BrowserRouter>
    )
  }
}
export default IRouter
```

在BrowserRouter这个组件的下面嵌套了App组件，为什么这里要嵌套一个App组件呢？因为我们希望所有打开的组件都是在这个App组件中进行呈现。例如登录组件，单击菜单项，打开的组件都是在这个组件中呈现。

对应的App组件的代码如下：

```
import React from 'react';
// import logo from './logo.svg';
import './App.css';
// import {Button} from 'antd'
// import 'antd/dist/antd.css'
function App(props) {
  return (
    <div>
      // 接收传递过来的组件内容
      {props.children}
    </div>
  );
}

export default App;
```

对应的在项目的index.js中启动的是router.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
// 导入路由
import Router from './router'
import * as serviceWorker from './serviceWorker';
ReactDOM.render(<Router />, document.getElementById('root'));
```

对应的在项目创建一个pagecomponent目录

在该目录下创建一个login文件夹，对应在该文件夹下创建index.js文件，初步测试的代码如下：

```
import React, { Component } from 'react'
class Login extends Component{
  render(){
    return (
      <div>
        登录页面
      </div>
    )
  }
}
export default Login
```

2.2 嵌套路由设置

根据前面的路由设定，我们知道，输入地址：<http://localhost:3000/admin>

会打开后台页面，那么当单击该页面中的菜单项的时候，也会展示出对应的组件内容，那么这块内容对应的路由应该怎样进行处理呢？

这里单击菜单项，打开对应的组件内容，但是要注意打开的组件内容还是在后台页面中进行展示的，也就是Admin这个组件的内容还是需要继续展示的，所以对应的路由配置如下：

```
import React, { Component } from 'react'
import {BrowserRouter,Route} from 'react-router-dom'
import App from './App'
import Admin from './admin'
import Login from './pagecomponent/login'
import Buttons from './pagecomponent/buttons'
class IRouter extends Component{
  render(){
    return (
      <BrowserRouter>
        <App>
          <Route path="/login" component={Login} />
          { /* <Route path='/admin' component={Admin} /> */ }
          { /* 重新配置了Admin组件对应的路由 */ }
          <Route path='/admin' render={()=>
            <Admin>
              <Route path='/admin/componet/buttons' component={Buttons}/>
            </Admin>
          } />
        </App>
      </BrowserRouter>
    )
  }
}
export default IRouter
```

基本的Buttons组件的内容如下：在pagecomponent目录下的buttons目录中。

```
import React, { Component } from 'react'
class Buttons extends Component{
  render(){
    return (
      <div>
        登录按钮
      </div>
    )
  }
}
export default Buttons
```

在上面的代码中，我们在Admin这个组件中继续加载别的组件的内容，例如输入的地址为 (/admin/componet/buttons，这个地址在config/menuConfig.js中进行了配置)，这时会加载Button这个组件的内容，那么Button这个组件会在Admin这个组件中展示出来，那么在Admin这个组件的哪个位置进行展示呢？

回到src/admin.js文件

是在这一行中进行展示。

所以修改后的代码如下：

```
import React, { Component } from 'react';
import {Row,Col} from 'antd'
import Header from './components/Header'
import Footer from './components/Footer'
import NavLeft from './components/NavLeft'
import Home from './home'
import './style/common.css'
class Admin extends Component{
  render(){
    return (
      <Row className='container'>
        <Col span={3} className="nav-left">
          <NavLeft/>
        </Col>
        <Col span={21} className="main">
          <Header>

          </Header>
          { /*展示对应子组件内容*/ }
          <Row className="content">
            { /* <Home/> */ }
            {this.props.children}
          </Row>
          <Footer></Footer>
        </Col>
      </Row>
    )
  }
}
export default Admin
```

下面给所有的菜单项，添加对应的链接，回到src/components/NavLeft目录

```
//加载菜单,通过递归的方式
loadMenu=(data)=>{
  return data.map((item)=>{
    if(item.children){
      return(
        <SubMenu title={item.title} key={item.key}>
          {this.loadMenu(item.children)}
        </SubMenu>
      )
    }
    return <Menu.Item title={item.title} key={item.key}>
      { /* 给所有的菜单项加上对应的链接 */ }
      <Link to={item.key}>{item.title}</Link>
    </Menu.Item>
  })
}
```

在上面的代码中，给所有的菜单项添加了对应的链接。

3、基础组件使用

3.1 按钮组件使用

基本的按钮如下，分别为 主按钮、次按钮、虚线按钮、危险按钮和链接按钮。

```
<Button type="primary">按钮</Button>
<Button>按钮</Button>
<Button type="dashed">按钮</Button>
<Button type="danger">按钮</Button>
<Button type="link">按钮</Button>
```

上面创建的都是基本的按钮，下面看一下图形按钮。

[illegible]

控制正在加载的状态

```
state={
  isLoading:true
}
handleClick={()=>{
  this.setState({
    isLoading:false
  })
}
<Button onClick={this.handleClick}>关闭</Button>
```

完整代码

[illegible]

3.2 弹出窗口的使用

弹出的窗口地址: <https://ant.design/components/modal-cn/>

弹出窗口的实现

下面先实现一个弹出窗口

在pagecomponent目录下创建modals目录。

对应的代码如下：

```
import React, { Component } from 'react'
import {Button,Modal} from 'antd'
class Modals extends Component{
  state={
    showModal:false,
  }
  handleClick=()=>{
```

```

        this.setState({showModal:true})
    }
    render(){
        return (
            <div>
                { /* 创建按钮 */ }
                <Button type="primary" onClick={this.handleClick}>打开窗口</Button>
                { /* 创建模态窗口 */ }
                <Modal title="React课程" visible={this.state.showModal}
                    onCancel={()=>{
                        this.setState({showModal: false})
                    }}
                    onOk={()=>{
                        alert('你点击了确定按钮')
                        this.setState({showModal: false})
                    }}
                >
                    <p>Hello React</p>
                </Modal>
            </div>
        )
    }
}
export default Modals在

```

在这里需要注意的是，不要忘记添加路由

```

import Modals from './pagecomponent/modals'
<Route path='/admin/componet/modals' component={Modals}/>

```

下面可以改变弹出的窗口中按钮的文字

该窗口添加如下两个属性

```

okText='确定'
cancelText='取消'

```

创建‘确认窗口’

通过Modal.confirm这个API完成。7:29--7:58

```

// 确认窗口
handleConfirm={()=>{
    Modal.confirm({
        title:'删除提示',
        content:'你确定要删除吗?',
        onOk(){
            console.log('删除成功')
        },
        onCancel(){
            console.log('删除失败')
        }
    })
}}

<Button onClick={this.handleConfirm} type="primary">确认窗口</Button>

```

完整代码如下:

```

import React, { Component } from 'react'
import { Button,Modal} from 'antd'
class Modals extends Component{
    state={
        showModal:false,
    }
    // 打开窗口
    handleClick={()=>{
        this.setState({showModal:true})
    }}
    // 确认窗口
    handleConfirm={()=>{
        Modal.confirm({
            title:'删除提示',
            content:'你确定要删除吗?',
            onOk(){
                console.log('删除成功')
            },
            onCancel(){
                console.log('删除失败')
            }
        })
    }}
    render(){
        return (
            <div>

```

```

    /* 创建按钮 */
    <Button type="primary" onClick={this.handleClick}>打开窗口</Button>&nbsp;&nbsp;&nbsp;
    <Button onClick={this.handleConfirm} type="primary">确认窗口</Button>

    /* 创建模态窗口 */
    <Modal title="React课程" visible={this.state.showModal}
      onCancel={()=>{
        this.setState({showModal: false})
      }}
      onOk={()=>{
        alert('你点击了确定按钮')
        this.setState({showModal: false})
      }}
      okText='确定'
      cancelText='取消'
    >
      <p>Hello React</p>
    </Modal>
  </div>
)
}
}
export default Modals

```

7:51--9:14

还有其它一些API,如下所示

```

Modal.info

Modal.success

Modal.error

Modal.warning

```

3.3 通知信息提示框

下面看一下通知信息提示框的具体实现(/pagecomponent/notifications)

<https://ant.design/components/notification-cn/>

具体的代码如下

```

import React, { Component } from 'react'
import { Button, notification } from 'antd'
class Notification extends Component {
  handleClick={()=>{
    notification.success({
      message: '订单确认',
      description: '用户下订单了,请及时的确认',
      placement: 'bottomRight'
    })
  }}
  render() {
    return (
      <div>
        <Button type="primary" onClick={this.handleClick}>通知提醒</Button>
      </div>
    )
  }
}
export default Notification

```

注意：这里别忘记配置路由4:42--5:04

```

import Notification from './pagecomponent/notifications'
<Route path="/admin/componet/notification" component={Notification}></Route>

```

当然这里还有其它的一些API

```

notification.success(config)

notification.error(config)

notification.info(config)

notification.warning(config)

notification.warn(config)

notification.open(config)

```

```
notification.close(key: String)

notification.destroy()
```

3.4 Tabs标签页

基本Tabs标签页使用

<https://ant.design/components/tabs-cn/>

以下的代码参考文档，进行修改（pagecomponent/tabs）

```
import React, { Component } from 'react'
import { Tabs } from 'antd'
const { TabPane } = Tabs;
class Tab extends Component{
  callback=(key)=>{
    console.log('你单击的是第'+key+'个页签');
  }
  render(){
    return (
      <div>
        <Tabs defaultActiveKey="1" onChange={this.callback}>
          <TabPane tab="Tab 1" key="1">
            用户列表信息
          </TabPane>
          <TabPane tab="Tab 2" key="2">
            商品列表信息
          </TabPane>
          <TabPane tab="Tab 3" key="3">
            订单列表信息
          </TabPane>
        </Tabs>
      </div>
    )
  }
}
export default Tab
```

注意：需要添加路由

给Tab页签添加对应的图标(参考文档)。

```
import { Tabs, Icon } from 'antd'
<TabPane tab={<span><Icon type="apple" />Tab 1</span>} key="1">
  用户列表信息
</TabPane>
```

动态Tabs标签也实现*

可以实现页签的添加 与删除，这个例子文档中有详细的代码，可以自行参考学习。

4、Form表单使用

4.1 登录表单处理

下面先创建一个基本的登录表单

```
import React,{Component} from 'react'
import {Form,Input,Button} from 'antd'

class FormLogin extends Component{
  render(){
    return (
      <div>
        <Form style={{width:300}}>
          <Form.Item>
            <Input placeholder="请输入用户名"></Input>
          </Form.Item>
          <Form.Item>
            <Input placeholder="请输入密码"></Input>
          </Form.Item>
          <Form.Item>
            <Button type="primary">登录</Button>
          </Form.Item>
        </Form>
      </div>
    )
  }
}
export default FormLogin
```

下面实现表单的校验，下面的代码给文本框中添加了默认值，以及对应的校验规则

```
import React,{Component} from 'react'
import {Form,Input,Button} from 'antd'

class FormLogin extends Component{
  render(){
    //获取getFieldDecorator,在这个方法中定义校验的规则
    const {getFieldDecorator}=this.props.form;
    return (
      <div>
        <Form style={{width:300}}>
          <Form.Item>
            {
              // 定义校验规则以及初始值
              getFieldDecorator('userName',{
                initialValue:'zhangsan',
                rules:[{
                  required:true,
                  message:'用户名不能为空'
                }]
            })(<Input placeholder="请输入用户名"></Input>)// 将文本框传递到方法中
          </Form.Item>
          <Form.Item>
            {
              getFieldDecorator('userPwd',{
                initialValue:'123456',
                rules:[{
                  required:true,
                  message:'密码不能为空'
                }]
            })(<Input placeholder="请输入密码" type="password"></Input>)
          </Form.Item>
          <Form.Item>
            <Button type="primary">登录</Button>
          </Form.Item>
        </Form>
      </div>
    )
  }
}
export default Form.create()(FormLogin)//创建表单,这里必须这么写,才能后获取到getFieldDecorator
```

执行上面的代码，如果文本框中没有输入对应的值，会出现对应的错误提示。

下面实现单击“登录”按钮，来完成对应的校验。

```
import React,{Component} from 'react'
import {Form,Input,Button,message} from 'antd'

class FormLogin extends Component{
  // 单击登录按钮的时候开始进行校验
  handleSubmit=()=>{
    //获取表单中所有的值
    let users=this.props.form.getFieldsValue();
    //进行校验
    this.props.form.validateFields((err,values)=>{
      if(!err){
        message.success(`${users.userName} 登录成功,密码是:${values.userPwd}`)
      }
    })
  }
  render(){
    const {getFieldDecorator}=this.props.form;
    return (
      <div>
        <Form style={{width:300}}>
          <Form.Item>
            {
              getFieldDecorator('userName',{
                initialValue:'',
                rules:[{
                  required:true,
                  message:'用户名不能为空'
                }]
            })(<Input placeholder="请输入用户名"></Input>)
          </Form.Item>
        </Form>
      </div>
    )
  }
}
```



```

      <Form.Item>
        {
          getFieldDecorator('userPwd',{
            initialValue:'',
            rules:[{
              required:true,
              message:'密码不能为空'
            }]
          })(<Input placeholder="请输入密码" type="password"></Input>)
        }
      </Form.Item>
      <Form.Item>
        //添加单击事件
        <Button type="primary" onClick={this.handleSubmit}>登录</Button>
      </Form.Item>
    </Form>
  </div>
)
}
}
export default Form.create()(FormLogin)

```

下面继续完善表单的校验。在这里使用了正则表达式

```

import React,{Component} from 'react'
import {Form,Input,Button,message, Icon} from 'antd'

class FormLogin extends Component{
  // 单击登录按钮的时候开始进行校验
  handleSubmit=()=>{
    // 获取表单中所有的值
    let users=this.props.form.getFieldsValue();
    // 进行校验
    this.props.form.validateFields((err,values)=>{
      if(!err){
        message.success(`${users.userName}登录成功,密码是:${values.userPwd}`)
      }
    })
  }
  render(){
    const {getFieldDecorator}=this.props.form;
    return (
      <div>
        <Form style={{width:300}}>
          <Form.Item>
            {
              getFieldDecorator('userName',{
                initialValue:'',
                rules:[{
                  required:true,
                  message:'用户名不能为空'
                }],{//判断范围
                  min:3,
                  max:20,
                  message:'用户名长度不在指定范围内'
                }},{//正则表达式
                  pattern:/^\w/g,
                  message:'用户名必须为英文字母或者是数字'
                }
              ])
              //给文本框添加前缀图标
            }(<Input prefix={<Icon type='user'></Icon>} placeholder="请输入用户名"></Input>)
          </Form.Item>
          <Form.Item>
            {
              getFieldDecorator('userPwd',{
                initialValue:'',
                rules:[{
                  required:true,
                  message:'密码不能为空'
                }]
              })(<Input prefix={<Icon type='lock'></Icon>} placeholder="请输入密码" type="password">
            </Input>)
            }
          </Form.Item>
          <Form.Item>
            <Button type="primary" onClick={this.handleSubmit}>登录</Button>
          </Form.Item>
        </Form>
      </div>
    )
  }
}

```

```

    }
  }
  export default Form.create()(FormLogin)

```

下面是添加‘记住密码’复选框

```

import React,{Component} from 'react'
import {Form,Input,Button,message, Icon,Checkbox} from 'antd'

class FormLogin extends Component{
  // 单击登录按钮的时候开始进行校验
  handleSubmit=()=>{
    //获取表单中所有的值
    let users=this.props.form.getFieldsValue();
    //进行校验
    this.props.form.validateFields((err,values)=>{
      if(!err){
        message.success(`${users.userName}登录成功,密码是:${values.userPwd}`)
      }
    })
  }
  render(){
    const {getFieldDecorator}=this.props.form;
    return (
      <div>
        <Form style={{width:300}}>
          <Form.Item>
            {
              getFieldDecorator('userName',{
                initialValue:'',
                rules:[{
                  required:true,
                  message:'用户名不能为空'
                },{
                  min:3,
                  max:20,
                  message:'用户名长度不在指定范围内'
                },{
                  pattern:/^\w/g,
                  message:'用户名必须为英文字母或者是数字'
                }]
              })(<Input prefix={<Icon type='user'></Icon>} placeholder="请输入用户名"></Input>)
            }
          </Form.Item>
          <Form.Item>
            {
              getFieldDecorator('userPwd',{
                initialValue:'',
                rules:[{
                  required:true,
                  message:'密码不能为空'
                }]
              })(<Input prefix={<Icon type='lock'></Icon>} placeholder="请输入密码" type="password">
            </Input>)
            }
          </Form.Item>
          {/* 增加 '记住密码' 复选框
            注意: 要想复选框默认选中, 需要将initialValue设置为true,同时前面需要添加'valuePropName'为'checked'才起
            作用
          */}
          <Form.Item>
            {
              getFieldDecorator('remember',{
                valuePropName:'checked',
                initialValue:true
              })(<Checkbox>记住密码</Checkbox>)
            }
          </Form.Item>
          <Form.Item>
            <Button type="primary" onClick={this.handleSubmit}>登录</Button>
          </Form.Item>
        </Form>
      </div>
    )
  }
}
export default Form.create()(FormLogin)

```

4.2 注册表单处理

<https://ant.design/components/grid-cn/>

先创建两个文本框，就是把登录表单中的用户名和密码两个文本框，拷贝过来。

```
import React, {Component} from 'react'
import {Form, Button, Input, Checkbox, Radio, Select, Switch, DatePicker, TimePicker, Upload, Icon, message} from 'antd'
class Register extends Component {
  render() {
    const {getFieldDecorator} = this.props.form;
    return (
      <div>
        <Form>
          <Form.Item label="用户名">
            {
              getFieldDecorator('userName', {
                initialValue: '',
                rules: [{
                  required: true,
                  message: '用户名不能为空'
                }, {
                  min: 3,
                  max: 20,
                  message: '用户名长度不在指定范围内'
                }, {
                  pattern: /\w/g,
                  message: '用户名必须为英文字母或者是数字'
                }]
              })(<Input prefix={<Icon type='user'></Icon>} placeholder="请输入用户名"></Input>)
            }
          </Form.Item>
          <Form.Item label="用户密码">
            {
              getFieldDecorator('userPwd', {
                initialValue: '',
                rules: [{
                  required: true,
                  message: '密码不能为空'
                }]
              })(<Input prefix={<Icon type='lock'></Icon>} placeholder="请输入密码" type="password">
            }
          </Form.Item>
        </Form>
      </div>
    )
  }
}
export default Form.create()(Register)
```

运行上面的代码以后，发现前面显示的文本内容，和文本框不在一行显示，那么这里需要用到栅格来进行布局。

<https://ant.design/components/grid-cn/>

4.2.1 栅格布局

```
import React, {Component} from 'react'
import {Form, Button, Input, Checkbox, Radio, Select, Switch, DatePicker, TimePicker, Upload, Icon, message} from 'antd'
class Register extends Component {
  render() {
    const {getFieldDecorator} = this.props.form;
    // 进行栅格布局
    const formItemLayout = {
      // labelCol: 表示标签占据的列。(可以在文档中进行查看Form表单的API)
      labelCol: {
        // xs: 表示屏幕 < 576px 时，使用xs栅格系统，占据24列，表示独占一行，也就是现在默认的情况
        xs: 24,
        sm: 4 // sm 表示屏幕 >= 576px 时，使用sm栅格系统，占据4列。
      },
      // 表示文本框
      wrapperCol: {
        xs: 24,
        sm: 12
      }
    }
    return (
      <div>
        <Form>
          <Form.Item label="用户名" {...formItemLayout}>
            {
              getFieldDecorator('userName', {
                initialValue: '',
                rules: [{
                  required: true,

```

```

        message: '用户名不能为空'
      }, {
        min: 3,
        max: 20,
        message: '用户名长度不在指定范围内'
      }, {
        pattern: /\w/g,
        message: '用户名必须为英文字母或者是数字'
      }
    ]
  })(<Input prefix={<Icon type='user'></Icon>} placeholder="请输入用户名"></Input>)
}
</Form.Item>
<Form.Item label="用户密码" {...formItemLayout}>
  {
    getFieldDecorator('userPwd', {
      initialValue: '',
      rules: [{
        required: true,
        message: '密码不能为空'
      }]
    })(<Input prefix={<Icon type='lock'></Icon>} placeholder="请输入密码" type="password">
</Input>)
  }
</Form.Item>
</Form>
</div>
)
}
}

```

在上面的代码中，使用了formItemLayout这个对象，来控制响应式布局。然后将该对象应用到Form.Item这个组件中，注意这里进行解构使用。

4.2.2 单选按钮与数字输入框

```

{ /* 性别 */
  <Form.Item label="性别" {...formItemLayout}>
    {
      getFieldDecorator('sex', {
        initialValue: '1',

      })(<Radio.Group>
        <Radio value="1">男</Radio>
        <Radio value="2">女</Radio>
      </Radio.Group>)
    }
  </Form.Item>
  { /* 年龄, 在这里主要使用的是InputNumber, 数字输入框 */
  <Form.Item label="年龄" {...formItemLayout}>
    {
      getFieldDecorator('age', {
        initialValue: 20,

      })(<InputNumber/>)
    }
  </Form.Item>
}

```

4:18--4:58

4.2.3 下拉框的实现


```

const { Option } = Select;
{ /* 下拉框 */
  <Form.Item label="所在城市" {...formItemLayout}>
    {
      getFieldDecorator('city', {
        initialValue: '3', // 默认选中'深圳'

      })(
        <Select>
          <Option value='1'>北京</Option>
          <Option value='2'>上海</Option>
          <Option value='3'>深圳</Option>
        </Select>
      )
    }
  </Form.Item>
}

```

```
</Form.Item>
```

下拉框使用的是 

```
<Form.Item label="兴趣爱好" {...formItemLayout}>
  {
    getFieldDecorator('love',{
      initialValue:['2','3'],

    })(
      <Select mode="multiple">
        <Option value='1'>吃饭</Option>
        <Option value='2'>睡觉</Option>
        <Option value='3'>外游戏</Option>
      </Select>
    )
  }
</Form.Item>
```

这里需要注意的是，给Select添加了一个mode属性，该属性的取值为'multiple'。

同时，初始值initialValue属性，可以为数组，定义的初始值为两个。

4.2.4 日期组件与TextArea组件使用

```
npm install moment --save
```

地址：<http://momentjs.cn/>

日期组件的使用，具体代码如下：

```
import moment from 'moment' // 导入moment日期处理库
<Form.Item label="出生日期" {...formItemLayout}>
  {
    getFieldDecorator('birthday',{
      initialValue:moment('2019-11-12'),

    })(
      <DatePicker/>
    )
  }
</Form.Item>
```

在进行初始化的时候，需要用到moment这个日期处理库，可以进行下载。

TextArea组件基本使用如下：

```
<Form.Item label="个人介绍" {...formItemLayout}>
  {
    getFieldDecorator('content',{
      initialValue: '请填写个人介绍'

    })(
      <Input.TextArea
        autoSize={
          {
            minRows:4,
            maxRows:6
          }
        }
      />
    )
  }
</Form.Item>
```

4.2.5 注册按钮实现

```
{/* 注册按钮 */}
<Form.Item {...buttonItemLayout}>
  {
    getFieldDecorator('btnRegister')(
      <Button type="primary">注册</Button>
    )
  }
</Form.Item>
```

注册按钮的布局代码：

```
const buttonItemLayout = {
  wrapperCol:{
    xs:24,
    sm:{
      span:12, //等价sm:12
      offset:4 //向右偏移4列，因为其他表单元素前面有文本，占据了4列，而按钮没有，所有向右偏移4列。
    }
  }
}
```

5、用户信息管理

5.1 简单表格使用

```
import React,{Component} from 'react';
import {Table} from 'antd'
class BaseTable extends Component{
  state = {}
  // 定义数据源
  componentDidMount(){
    const dataSource=[
      {
        id:1,
        userName: '张三',
        sex:1,
        state:1,
        love:1,
        birthday: '2012-1-2',
        content: '是一个大帅哥'
      },
      {
        id:2,
        userName: '李四',
        sex:1,
        state:1,
        love:1,
        birthday: '2012-1-2',
        content: '是一个大帅哥'
      },
      {
        id:3,
        userName: '王五',
        sex:1,
        state:1,
        love:1,
        birthday: '2012-1-2',
        content: '是一个大帅哥'
      }
    ]
    this.setState({dataSource})
  }
  render(){
    // 定义表头
    const columns=[{
      title: '编号', // 表头文本
      dataIndex: 'id' // 服务端返回的数据的key
    }, {
      title: '用户名',
      dataIndex: 'userName'
    }, {
      title: '性别',
      dataIndex: 'sex'
    }, {
      title: '状态',
      dataIndex: 'state'
    }, {
      title: '爱好',
      dataIndex: 'love'
    }, {
      title: '出生日期',
      dataIndex: 'birthday'
    }, {
      title: '个人介绍',
      dataIndex: 'content'
    }
    ]
    return (
      <div>
        <Table rowKey="id" columns={columns} dataSource={this.state.dataSource} bordered/>
      </div>
    )
  }
}
export default BaseTable
```

注意：别忘记定义路由

5.2 Mock数据处理

在上一个案例中，我们表格中展示的数据，是写死的，但是在企业开发中，数据肯定是不能写死的。

而是从服务端获取到的，服务端开发人员会开发对应的API接口地址，我们调用该地址，获取对应的数据。

那么，如果在开发中，服务端人员还没有开放对应的API，那么我们可以通过Mock来模拟一组数据，来进行远程的调用测试。

Mock语法规则使用可以参考：<https://github.com/nuysoft/Mock/wiki>。

为了能够快速的搭建Mock的使用环境，可以使用如下平台：

<https://www.easy-mock.com/login> (laowang1234567 123321) (该网站不稳定，这次没有使用)

学习网站：<https://www.cnblogs.com/samsara-yx/p/10882703.html>

在该平台中可以创建项目，并且创建对应的接口。

```
{
  "code": 0,
  "msg": '',
  "result": [{ // 表示产生10条数据
    "id": 1,
    "userName": '@cname',
    "sex": 1,
    "state": 1,
    "love": 1,
    "birthday": '2012-1-2',
    "content": '是一个大帅哥'
  }],
  "page": 1,
  "page_size": 10,
  "total_count": 60
}
```

"sex": 1 表示的是sex性别默认值为1，数据的范围在1-2之间随机产生。

上面定义的都是Mock的语法。

在项目中，想获取Mock产生的数据，可以通过axios来完成，下面说一下关于axios的安装

```
npm install axios --save
```

具体获取服务端的数据

这里重新定义一个表格，表头和第一个表格一样，获取的数据来源为

```
state = {
  dataSource2: []
}
```

dataSource2这个数组中的数据，来源于远程服务器，所以在componentDidMount方法中发起ajax请求

```
// 调用request方法，获取远程数据
this.request();
// request方法的具体定义实现如下
// 通过axios获取远程服务器数据
request={() => {
  let baseUrl = 'http://localhost:8081'
  axios.get(baseUrl + '/api/getUserList').then((res) => {
    if (res.status === 200) {
      this.setState({
        dataSource2: res.data.message
      })
    }
  })
}}
}
```

在render方法中定义表格

```
<Table rowKey="id" columns={columns} dataSource={this.state.dataSource2} bordered/>
```

完整代码如下：

```
import React, { Component } from 'react';
import { Table } from 'antd';
import axios from 'axios';
class BaseTable extends Component {
  state = {
    dataSource2: []
  }
  // 定义数据源
  componentDidMount() {
    const dataSource = [
      {
        id: 1,
        userName: '张三',
        sex: 1,
        state: 1,
        love: 1,
        birthday: '2012-1-2',
      }
    ]
  }
}
```



```

        content: '是一个大帅哥'
      },
      {
        id: 2,
        userName: '李四',
        sex: 1,
        state: 1,
        love: 1,
        birthday: '2012-1-2',
        content: '是一个大帅哥'
      },
      {
        id: 3,
        userName: '王五',
        sex: 1,
        state: 1,
        love: 1,
        birthday: '2012-1-2',
        content: '是一个大帅哥'
      }
    ]
    this.setState({dataSource})
    // 调用request方法, 获取远程数据
    this.request();
  }

// 通过axios获取远程服务器数据
request={()=>{
  let baseUrl='http://localhost:8081'
  axios.get(baseUrl+'/api/getUserList').then((res)=>{
    if(res.status===200){
      this.setState({
        dataSource2: res.data.message
      })
    }
  })
}}
}

render(){
  // 定义表头
  const columns=[
    {
      title: '编号', // 表头文本
      dataIndex: 'id' // 服务端返回的数据的key
    },
    {
      title: '用户名',
      dataIndex: 'userName'
    },
    {
      title: '性别',
      dataIndex: 'sex'
    },
    {
      title: '状态',
      dataIndex: 'state'
    },
    {
      title: '爱好',
      dataIndex: 'love'
    },
    {
      title: '出生日期',
      dataIndex: 'birthday'
    },
    {
      title: '个人介绍',
      dataIndex: 'content'
    }
  ]

  return (
    <div>
      <Table rowKey="id" columns={columns} dataSource={this.state.dataSource} bordered/>
      <hr/>
      <Table rowKey="id" columns={columns} dataSource={this.state.dataSource2} bordered/>
    </div>
  )
}
}
export default BaseTable

```

5.3 axios封装处理

axios详细配置: <http://www.axios-js.com/zh-cn/docs/#axios-post-url-data-config-1>

在src目录下创建一个axios目录, 对应的创建一个index.js文件, 在该文件中封装了关于axios的内容

```

import axios from 'axios';
export default class Axios{
  static ajaxAxios(options){
    const baseApi='http://localhost:8081/api';

```

```

    return new Promise((resolve, reject) => {
      axios({
        url: options.url,
        method: 'get',
        baseURL: baseApi,
        timeout: 5000,
        params: (options.data && options.data.params) || ''
      }).then((response) => {
        if (response.status === 200) {
          let res = response.data
          resolve(res)
        } else {
          reject(response)
        }
      })
    })
  }
}

```

那么对应的basicTable.js文件中的代码修改如下:

```

import axios from '../axios/index'

// 通过axios获取远程服务器数据
request() => {
  axios.ajaxAxios({
    url: '/getUserList',
    data: {
      params: {
        pageIndex: 1
      }
    }
  }).then((res) => {
    this.setState({
      dataSource2: res.message // 返回的是res.data, 所以这里直接接收message
    })
  })
  // let baseUrl = 'http://localhost:8081'
  // axios.get(baseUrl + '/api/getUserList').then((res) => {
  //   if (res.status === 200) {
  //     this.setState({
  //       dataSource2: res.data.message
  //     })
  //   }
  // })
}

```

最后将表格中展示的数据, 处理一下, 这里是将显示的数字, 用具体的文本内容进行替换。

```

const columns = [
  {
    title: '编号', // 表头文本
    dataIndex: 'id' // 服务端返回的数据的key
  }, {
    title: '用户名',
    dataIndex: 'userName'
  }, {
    title: '性别',
    dataIndex: 'sex',
    render(sex) {
      return sex === 1 ? '男' : '女'
    }
  }, {
    title: '状态',
    dataIndex: 'state',
    render(state) {
      let config = {
        '1': '已经查看',
        '2': '已经审批',
        '3': '已经编辑'
      }
      return config[state]
    }
  }, {
    title: '爱好',
    dataIndex: 'love',
    render(love) {
      let config = {
        '1': '吃饭',
        '2': '睡觉',
        '3': '玩游戏'
      }
    }
  }
]

```

```

        return config[love];
    }
  }, {
    title: '出生日期',
    dataIndex: 'birthday'
  }, {
    title: '个人介绍',
    dataIndex: 'content'
  }
]

```

5.4 获取选中用户信息

为了获取选中的用户信息，可以在每行的前面添加一个单选按钮，具体的添加方式，如下先添加一个Table表格，然后给其添加一个属性rowSelection，表示行选中的方式（一般通过单选按钮和复选框），它的取值为对象。

```

<Table
  rowKey="id"
  loading={this.state.loading}
  columns={columns}
  dataSource={this.state.dataSource2}
  bordered
  // 添加单选按钮
  rowSelection={rowSelection}
  // 单击行中任意单元格，选中单选按钮
  // onRow: 设置行属性, 用来设置单击某一行
  onRow={(record, index) => {
    return {
      onClick: () => {
        this.onRowClick(record, index)
      }
    }
  }}
/>

```

对应的rowSelection具体的定义方式如下，在render方法中，具体定义如下

```

// 定义状态
const rowSelection={
  type: 'radio',
  // 获取选中行对应的索引，让对应的单选按钮选中
  selectedRowKeys: this.state.selectedRowKeys
}

```

type: 表示选择的类型，这里是单选按钮。

selectedRowKeys: 获取选中行的key值，也就是获取选中行的对应的索引。这里从state中获取，那么怎样给state中保存选中的行的索引值呢。需要给Table控件，添加一个onRow属性，该属性表示单击某一行，对应的返回一个方法，这里可以获取选中的行的记录信息，以及对应的索引值，并且给所单击的行添加了一个单击事件，执行onRowClick这个方法，把行的索引以及记录传递到该方法中，（这样就可以将选中的行的索引值给了selectedRowKeys）那么该方法的定义如下：（在render方法上面定义）

```

// 选中行处理
onRowClick=(record, index) => {
  let selectKey=[index+1];
  Modal.info({
    title: '展示用户信息',
    content: `用户名: ${record.userName}`
  })
  this.setState({
    // selectedRowKeys: 指定选中项的 key值，是一个数组
    selectedRowKeys: selectKey,
    selectedItem: record // 把记录保存到state中的selectedItem中，发现以后操作。
  })
}

```

index的值从0开始计算，所以这里需要加1，同时将selectKey(selectKey是一个数组，因为selectedRowKeys是一个数组)赋值给selectedRowKeys，同时将选中的行的数据保存到selectedItem中。

5.5 批量删除用户信息

首先增加一个table表格

```

<Table
  rowKey="id"
  loading={this.state.loading}
  columns={columns}
  dataSource={this.state.dataSource2}
  bordered
  // 添加单选按钮
  rowSelection={rowCheckSelection}
  // 单击行中任意单元格，选中单选按钮
  // onRow: 设置行属性, 用来设置单击某一行
  onRow={(record, index) => {

```

```

        return {
          onClick: () => {
            this.onRowClick(record, index)
          }
        }
      }
    }
  }
}

```

这里需要修改一下rowSelection属性，将其行选择的类型修改成复选框

```

//定义复选框状态
const rowCheckSelection={
  type: 'checkbox',
  selectedRowKeys:this.state.selectedRowKeys,
  onChange: (selectedRowKeys,selectedRows)=>{
    //复选框状态改变的时候，会触发onChange(selectedRowKeys:选中行的索引, selectedRows: 选中行的对象,存储了选中行的数据)
    // console.log('selectedRowKeys=',selectedRowKeys);
    // console.log('selectedRows=',selectedRows);
    // 同时将这两个信息存储到state中，方便以后的操作
    this.setState({
      selectedRowKeys,
      selectedRows
    })
  }
}

```

添加一个按钮，将选中的用户信息删除掉。

```

<Button type="primary" onClick={this.handleClick}>删除</Button>
// 批量删除用户信息
handleClick={()=>{
  //从state中获取选中的行的信息
  let rows=this.state.selectedRows;
  let ids=[];
  rows.map((item)=>{
    ids.push(item.id)
  })
  Modal.confirm({
    title: '删除提示',
    content: `确定要删除选中的数据吗?${ids.join(',')}`,
    onOk: ()=>{
      message.success('删除成功!!')
    }
  })
}
}

```

5.6 分页封装实现

关于分页的处理，文档中对应的参数

<https://ant.design/components/pagination-cn/>

在这里，我们将其这些有关分页的参数进行了相应的封装处理。

由于这块内容在很多的页面中都会用到，所以这里将其进行封装处理。

在src目录下，创建一个utils目录，在该目录下创建utils.js文件，代码如下：

```

export default{
  //当切换页码时，调用callback
  pagination(data,callback){
    // https://ant.design/components/pagination-cn/
    // 创建对象，封装分页的信息
    return {
      //切换页码时调用该方法，current表示当前页码
      // 页码改变的回调，参数是改变后的页码及每页条数
      onChange: (current)=>{
        callback(current)
      },
      current:data.page,
      pageSize:data.page_size, //每页条数
      total:data.total, // 数据总数
      showTotal: ()=>{
        return `共${data.total}条`
      },
      // 是否可以快速跳转至某页（在页面上展示出对应的goto按钮）
      showQuickJumper:true
    }
  }
}

```

创建好以后，具体使用的方式如下：

在页面中增加了一个表格

```
<Table
  rowKey="id"
  loading={this.state.loading}
  columns={columns}
  dataSource={this.state.dataSource2}
  bordered
  // 添加复选框按钮
  rowSelection={rowCheckSelection}
  // 启用分页
  pagination={this.state.pagination}
  // 单击行中任意单元格, 选中单选按钮
  // onRow: 设置行属性, 用来设置单击某一行
  onRow={ (record, index) => {
    return {
      onClick: () => {
        this.onRowClick(record, index)
      }
    }
  }}
/>
```

在上面的代码中, 重点是添加了

```
pagination={this.state.pagination}
```

在这里我们是在state中完成, 对应分页方法的调用的, state中的代码如下

```
import Utils from '../utils/utils.js'
// 通过axios获取远程服务器数据
request={() => {
  this.setState({ loading: true });
  axios.ajaxAxios({
    url: '/getUserList',
    data: {
      params: {
        pageIndex: 1
      }
    }
  }).then((res) => {
    this.setState({
      dataSource2: res.message,
      loading: false,
      // 保存返回的分页有关的数据。
      // 切换页码的时候, 执行回调函数, 并且将当前页码作为参数传递过来。
      pagination: Utils.pagination(res, (current) => {

    })
  })
})
}
```

通过上面的代码, 可以发现我们是在request方法中调用封装好的分页的方法, 也就是将从服务端获取到的数据, 传递到方法中, 同时将返回的数据存储到state中。

通过上面的代码已经展示出对应的分页信息了, 下面要实现的就是分页的跳转了。

首先定义一个params对象, 该对象中封装了当前页码这个参数。

```
// 传递参数, 最开始传递的是第一页。
params={page: 1}
```

那么每次发送请求, 都要获取该参数

```
request={() => {
  let that = this; // 保存一下this
  this.setState({ loading: true });
  axios.ajaxAxios({
    url: '/getUserList',
    data: {
      params: {
        // pageIndex: 1
        page: this.params.page // 发送请求的时候传递参数。
      }
    }
  }).then((res) => {
    this.setState({
      dataSource2: res.message,
      loading: false,
      // 保存返回的分页有关的数据。
      pagination: Utils.pagination(res, (current) => {
```

```

        //把回传回来的当前页码给了page这个参数，同时重新发送请求。
        that.params.page=current;
        this.request();
    })
  })
}

```

这里执行上面的代码，可以发现每次切换页码的时候都会发送请求，但是没有实现数据的切换，主要是服务端没有实现。

整个完整代码如下：

```

import React,{Component} from 'react';
import {Table,Modal,Button,message} from 'antd'
// import axios from 'axios'
import axios from '../././axios/index'
import Utils from '../././utils/utils.js'
class BaseTable extends Component{
  state = {
    dataSource2:[],
    loading: false
  }
  //传递参数，最开始传递的是第一页。
  params={page:1}
  // 定义数据源
  componentDidMount(){
    const dataSource=[
      {
        id:1,
        userName: '张三',
        sex:1,
        state:1,
        love:1,
        birthday: '2012-1-2',
        content: '是一个大帅哥'
      },
      {
        id:2,
        userName: '李四',
        sex:1,
        state:1,
        love:1,
        birthday: '2012-1-2',
        content: '是一个大帅哥'
      },
      {
        id:3,
        userName: '王五',
        sex:1,
        state:1,
        love:1,
        birthday: '2012-1-2',
        content: '是一个大帅哥'
      }
    ]
    this.setState({dataSource})
    // 调用request方法，获取远程数据
    this.request();
  }

  // 通过axios获取远程服务器数据
  request=()=>{
    let that=this;
    this.setState({ loading: true });
    axios.ajaxAxios({
      url: '/getUserList',
      data:{
        params:{
          // pageIndex:1
          page:that.params.page
        }
      }
    }).then((res)=>{
      this.setState({
        dataSource2:res.message,
        loading:false,
        // 保存返回的分页有关的数据。
        pagination:Utils.pagination(res,(current)=>{
          that.params.page=current;
          this.request();
        })
      })
    })
  }
}

```

```

    })
    // let baseUrl='http://localhost:8081'
    // axios.get(baseUrl+'api/getUserList').then((res)=>{
    //     if(res.status===200){
    //         this.setState({
    //             dataSource2:res.data.message
    //         })
    //     }
    // })
    // })
    // })
}

```

//选中行处理

```

onRowClick=(record,index) => {
    let selectKey=[index+1];
    Modal.info({
        title:'展示用户信息',
        content:`用户名:${record.userName}`
    })
    this.setState({
        //selectedRowKeys:指定选中项的 key值，是一个数组
        selectedRowKeys:selectKey,
        selectedItem:record
    })
}

```

// 批量删除用户信息

```

handleClick=()=>{
    //从state中获取选中的行的信息
    let rows=this.state.selectedRows;
    let ids=[];
    rows.map((item)=>{
        ids.push(item.id)
    })
    Modal.confirm({
        title:'删除提示',
        content:`确定要删除选中的数据吗?${ids.join(',')}`,
        onOk:()=>{
            message.success('删除成功!!!')
        }
    })
}

```

render(){

// 定义表头

```

    const columns=[
        {
            title:'编号', // 表头文本
            dataIndex:'id' // 服务端返回的数据的key
        },{
            title:'用户名',
            dataIndex:'userName'
        },{
            title:'性别',
            dataIndex:'sex',
            render(sex){
                return sex===1?'男':'女'
            }
        },{
            title:'状态',
            dataIndex:'state',
            render(state){
                let config ={
                    '1':'已经查看',
                    '2':'已经审批',
                    '3':'已经编辑'
                }
                return config[state];
            }
        },{
            title:'爱好',
            dataIndex:'love',
            render(love){
                let config ={
                    '1':'吃饭',
                    '2':'睡觉',
                    '3':'玩游戏'
                }
                return config[love];
            }
        },{
            title:'出生日期',
            dataIndex:'birthday'
        },{
            title:'个人介绍',
            dataIndex:'content'
        }
    ]
}

```

// 定义单选按钮状态

```

const rowSelection={
    type:'radio',

```

```

//获取选中行对应的索引
selectedRowKeys:this.state.selectedRowKeys
}
//定义复选框状态
const rowCheckSelection={
  type: 'checkbox',
  selectedRowKeys:this.state.selectedRowKeys,
  onChange:(selectedRowKeys,selectedRows)=>{
    //复选框状态改变的时候，会触发onChange(selectedRowKeys:选中行的索引, selectedRows: 选中行的对象,存储了选中行的数据)
    // console.log('selectedRowKeys=',selectedRowKeys);
    // console.log('selectedRows=',selectedRows);
    // 同时将这两个信息存储到state中，方便以后的操作
    this.setState({
      selectedRowKeys,
      selectedRows
    })
  }
}
}
return (
  <div>
    <Table rowKey="id" columns={columns} dataSource={this.state.dataSource} bordered/>
    <hr/>
    <Table rowKey="id" loading={this.state.loading} columns={columns} dataSource={this.state.dataSource2} bordered/>
    <hr/>
    { /* 展示单选按钮 */ }
    <Table
      rowKey="id"
      loading={this.state.loading}
      columns={columns}
      dataSource={this.state.dataSource2}
      bordered
      // 添加单选按钮
      rowSelection={rowSelection}
      // 单击行中任意单元格，选中单选按钮
      // onRow: 设置行属性,用来设置单击某一行
      onRow={(record,index) =>{
        return {
          onClick: ()=>{
            this.onRowClick(record,index)
          }
        }
      }}
    />

    { /* 批量删除用户信息 */ }
    <Button type="primary" onClick={this.handleClick}>删除</Button>
    <Table
      rowKey="id"
      loading={this.state.loading}
      columns={columns}
      dataSource={this.state.dataSource2}
      bordered
      // 添加复选框按钮
      rowSelection={rowCheckSelection}
      // 单击行中任意单元格，选中单选按钮
      // onRow: 设置行属性,用来设置单击某一行
      onRow={(record,index) =>{
        return {
          onClick: ()=>{
            this.onRowClick(record,index)
          }
        }
      }}
    />

    { /* 分页实现 */ }
    <Table
      rowKey="id"
      loading={this.state.loading}
      columns={columns}
      dataSource={this.state.dataSource2}
      bordered
      // 添加复选框按钮
      rowSelection={rowCheckSelection}
      // 启用分页
      pagination={this.state.pagination}
      // 单击行中任意单元格，选中单选按钮
      // onRow: 设置行属性,用来设置单击某一行
      onRow={(record,index) =>{
        return {
          onClick: ()=>{
            this.onRowClick(record,index)
          }
        }
      }}
    />
  </div>
)

```



```

        />
      </div>
    )
  }
}
export default BaseTable

```

如果表格中的数据非常多，为了方便浏览，可以让表格的表头固定住。

固定表格的表头非常简单，加一个scroll属性

```
scroll={{y:220}}
```

设置纵向滚动，也可用于指定滚动区域的宽和高

5.7 删除单条记录

首先增加一个操作列。

```

{
  title: '操作',
  render: (item) => {
    return <Button onClick={() => this.handleDelete(item)}>删除</Button>
  }
}

```

对应的handleDelete方法实现如下：

```

// 删除一条记录
handleDelete=(item) => {
  let id = item.id;
  Modal.confirm({
    title: '确认',
    content: `您确定要删除${id}此记录吗?`,
    onOk: () => {
      message.success('删除成功')
    }
  })
}
}

```

6、商品管理

6.1 商品列表展示

```

import React, {Component} from 'react';
import {Table, Modal, message, Button} from 'antd'
import axios from '../../axios/index'
import Utils from '../../utils/utils.js'
class Product extends Component {
  state = {
    dataSource: [],
    loading: false,
    // 表示模态窗口
    showModal: false
  }
  params = {page: 1}
  componentDidMount() {
    this.request();
  }
  // 通过axios获取远程服务器数据
  request = () => {
    let that = this;
    this.setState({loading: true});
    axios.ajaxAxios({
      url: '/getProductList',
      data: {
        params: {
          // pageIndex: 1
          page: this.params.page
        }
      }
    }).then((res) => {
      this.setState({
        dataSource: res.message,
        loading: false,

```

```

        // 保存返回的分页有关的数据。
        pagination:Utils.pagination(res,(current)=>{
            that.params.page=current;
            this.request();
        })
    })
}
// 添加商品
handleAdd={()=>{
    this.setState({showModal:true})
}

render(){
    // 定义表头
    const columns=[{
        title:'编号', // 表头文本
        dataIndex:'id', // 服务端返回的数据的key

    },{
        title:'商品名称',
        dataIndex:'title',

    },{
        title:'商品图片',
        dataIndex:'img_url',
        render(img_url){
            return <img src={img_url}/>;
        }

    },{
        title:'市场价格',
        dataIndex:'market_price'

    },{
        title:'库存',
        dataIndex:'stock_quantity'

    },{
        title:'添加商品日期',
        dataIndex:'add_time'

    },{
        title:'单击次数',
        dataIndex:'click'

    },{
        title:'操作',
        // render:(item)=>{

        //     return <Button onClick={()=>this.handleDelete(item)}>删除</Button>

        // }

    ]
    //定义复选框状态
    const rowCheckSelection={
        type: 'checkbox',
        selectedRowKeys:this.state.selectedRowKeys,
        onChange:(selectedRowKeys,selectedRows)=>{
            //复选框状态改变的时候，会触发onChange(selectedRowKeys:选中行的索引, selectedRows: 选中行的对象,存储了选中行的数据)
            // console.log('selectedRowKeys=',selectedRowKeys);
            // console.log('selectedRows=',selectedRows);
            // 同时将这两个信息存储到state中，方便以后的操作
            this.setState({
                selectedRowKeys,
                selectedRows
            })
        }
    }
    return( <div>

        <Button type="primary" onClick={this.handleAdd}>添加商品</Button>
        <Table
            rowKey="id"
            loading={this.state.loading}
            columns={columns}
            dataSource={this.state.dataSource}
            bordered
            // 添加复选框按钮
            rowSelection={rowCheckSelection}
            // 启用分页
            pagination={this.state.pagination}
            // 单击行中任意单元格，选中单选按钮
            // onRow:设置行属性,用来设置单击某一行
            // onRow={(record,index) =>{
            //     return {
            //         onClick:()=>{

```

```

        //      this.onRowClick(record,index)
        //    }
        //  }
        // }}

    />
  </div>
}
}
export default Product

```

6.2 添加商品

在return 方法中，添加模态窗口,在模态窗口中添加表单，然后让窗口默认的隐藏。

```

{ /* 添加商品 */
  <Modal title="添加商品" visible={this.state.showModal}
    onCancel={()=>{
      this.setState({showModal: false})
    }}
    onOk={()=>{
      let products=this.props.form.getFieldsValue();
      //进行校验
      this.props.form.validateFields((err,values)=>{
        if(!err){
          axios.postAxios({
            url:'/addProduct',
            method:'post',
            data:products
          }).then((res)=>{
            message.success('商品添加成功!!')
            this.setState({showModal: false})
          })
        }
      })
    }}
  >
  <Form style={{width:300}}>
    <Form.Item>
      {
        // 定义校验规则以及初始值
        getFieldDecorator('title',{
          initialValue:'',
          rules:[{
            required:true,
            message:'商品名称不能为空'
          }]
        })(<Input placeholder="请输入商品名称"></Input>)// 将文本框传递到方法中
      }
    </Form.Item>
    <Form.Item>
      {
        // 定义校验规则以及初始值
        getFieldDecorator('market_price',{
          initialValue:'',
          rules:[{
            required:true,
            message:'商品市场价格不能为空'
          }]
        })(<Input placeholder="请输入商品市场价格"></Input>)// 将文本框传递到方法中
      }
    </Form.Item>
    <Form.Item>
      {
        // 定义校验规则以及初始值
        getFieldDecorator('sell_price',{
          initialValue:'',
          rules:[{
            required:true,
            message:'商品售卖价不能为空'
          }]
        })(<Input placeholder="请输入商品售卖价"></Input>)// 将文本框传递到方法中
      }
    </Form.Item>
    <Form.Item>

```

```

    {
      // 定义校验规则以及初始值
      getFieldDecorator('stock_quantity',{
        initialValue:'',
        rules:[{
          required:true,
          message:'商品库存不能为空'
        }]
      })(<Input placeholder="请输入商品库存"></Input>)// 将文本框传递到方法中
    }
  }
</Form.Item>
</Form>

```

子return方法的上面加上如下语句

```
const {getFieldDecorator}=this.props.form;
```

同时调用一下Form.create()方法

```
export default Form.create()(Product)
```

在页面中增加，添加按钮

```
<Button type="primary" onClick={this.handleClick}>添加商品</Button>
```

单击添加按钮，弹出口

```

// 添加商品
handleAdd={()=>{
  this.setState({showModal:true})
}}

```

在state中定义showModel这个状态

```

state = {
  dataSource:[],
  loading: false,
  //表示模态窗口
  showModal:false
}

```

当单击窗口中的‘确定’按钮的时候，通过axios发送请求，在这里发送的是post请求，对应的在axios目录下的index.js文件中，封装了一个处理post请求的方法。

```

import Qs from 'qs'
static ajaxPostAxios(options){
  const baseApi='http://localhost:8081/api';
  console.log('products=',options.data);
  return new Promise((resolve,reject) => {
    axios({
      url:options.url,
      method:options.method,
      baseURL:baseApi,
      timeout:5000,

      headers:{'Content-Type':'application/x-www-form-urlencoded'},
      //将对象 序列化URL的形式，以&进行拼接。(name=xxx&age=xxx)
      data:Qs.stringify(options.data),

    }).then((response)=>{
      if(response.status===200){
        let res=response.data
        resolve(res)
      }else{
        reject(response)
      }
    })
  })
}

```

6.3 删除商品信息

具体的做法与删除用户信息一样。