课程简介



ReactJS入门，Model层



Ant Design 入门



Ant Design Pro 入门

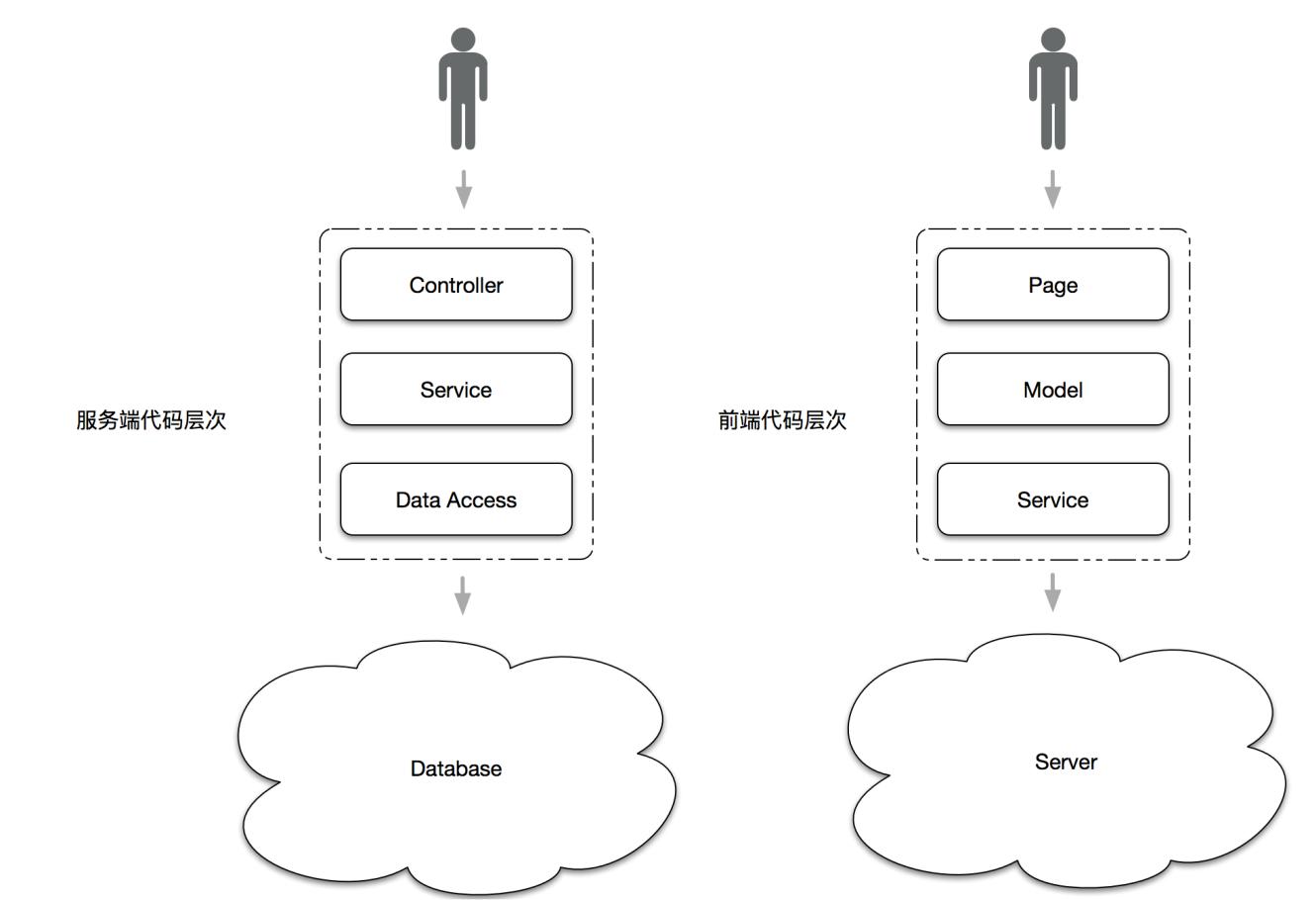


1、ReactJS入门



1.1、Model

1.1.1、分层



上图中，左侧是服务端代码的层次结构，由 Controller、Service、Data Access 三层组成服务端系统：

 Controller 层负责与用户直接打交道，渲染页面、提供接口等，侧重于展示型逻辑。

 Service 层负责处理业务逻辑，供 Controller 层调用。

 Data Access 层顾名思义，负责与数据源对接，进行纯粹的数据读写，供 Service 层调用。

上图的右侧是前端代码的结构，同样需要进行必要的分层：

 Page 负责与用户直接打交道：渲染页面、接受用户的操作输入，侧重于展示型交互

性逻辑。

 Model 负责处理业务逻辑，为 Page 做数据、状态的读写、变换、暂存等。

 Service 负责与 HTTP 接口对接，进行纯粹的数据读写。

1.1.2、使用DVA进行数据分层管理

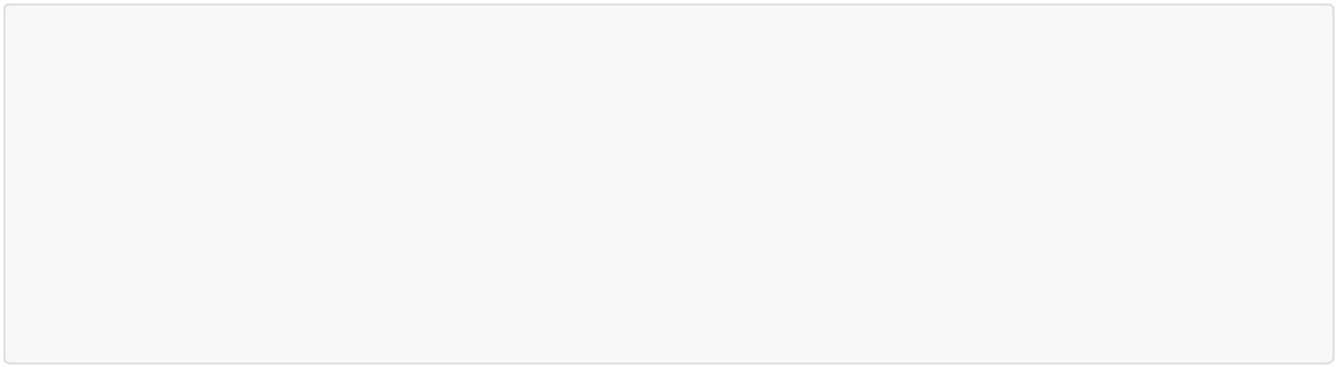
dva是基于 redux、redux-saga 和 react-router 的轻量级前端框架。官

网：<https://dvajs.com/>

对于dva我们不做过多详细的讲解，我们只要做到能够使用起来就可以了。对于想要全面学习dva框架的同学可自行研究。

首先，我们先将dva框架引入进来，由于umi对dva进行了整合，所以导入就变得非常简单了。

在config.js文件中进行配置：



export default {

plugins: [

['umi‐plugin‐react', {

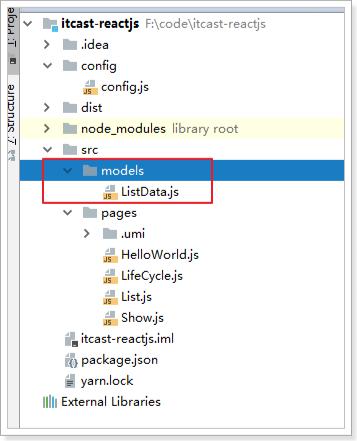
dva: true // 开启dva功能

}]

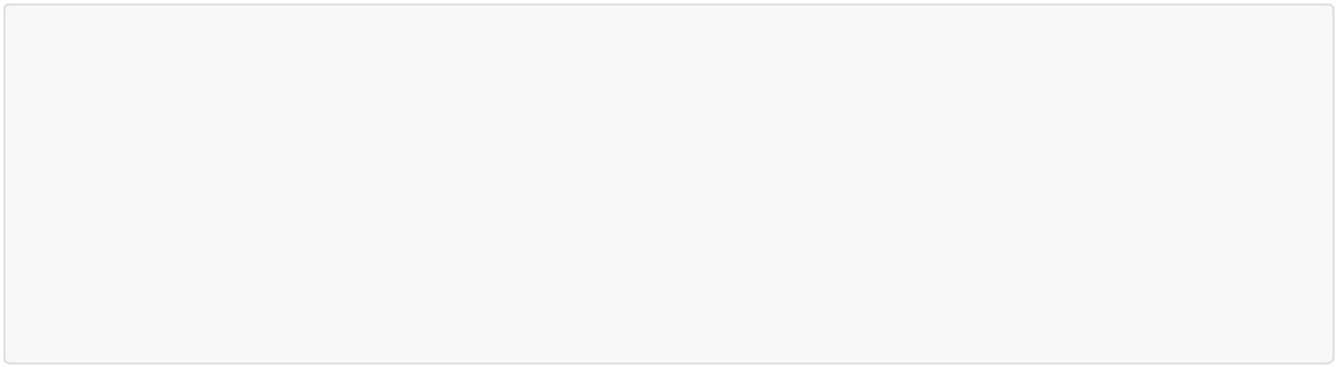
]

};

接下来，创建model文件，在umi中，约定在src/models文件夹中定义model，所以，在该文件夹下创建ListData.js文件：



编写内容：



export default {

namespace: 'list',

state: {

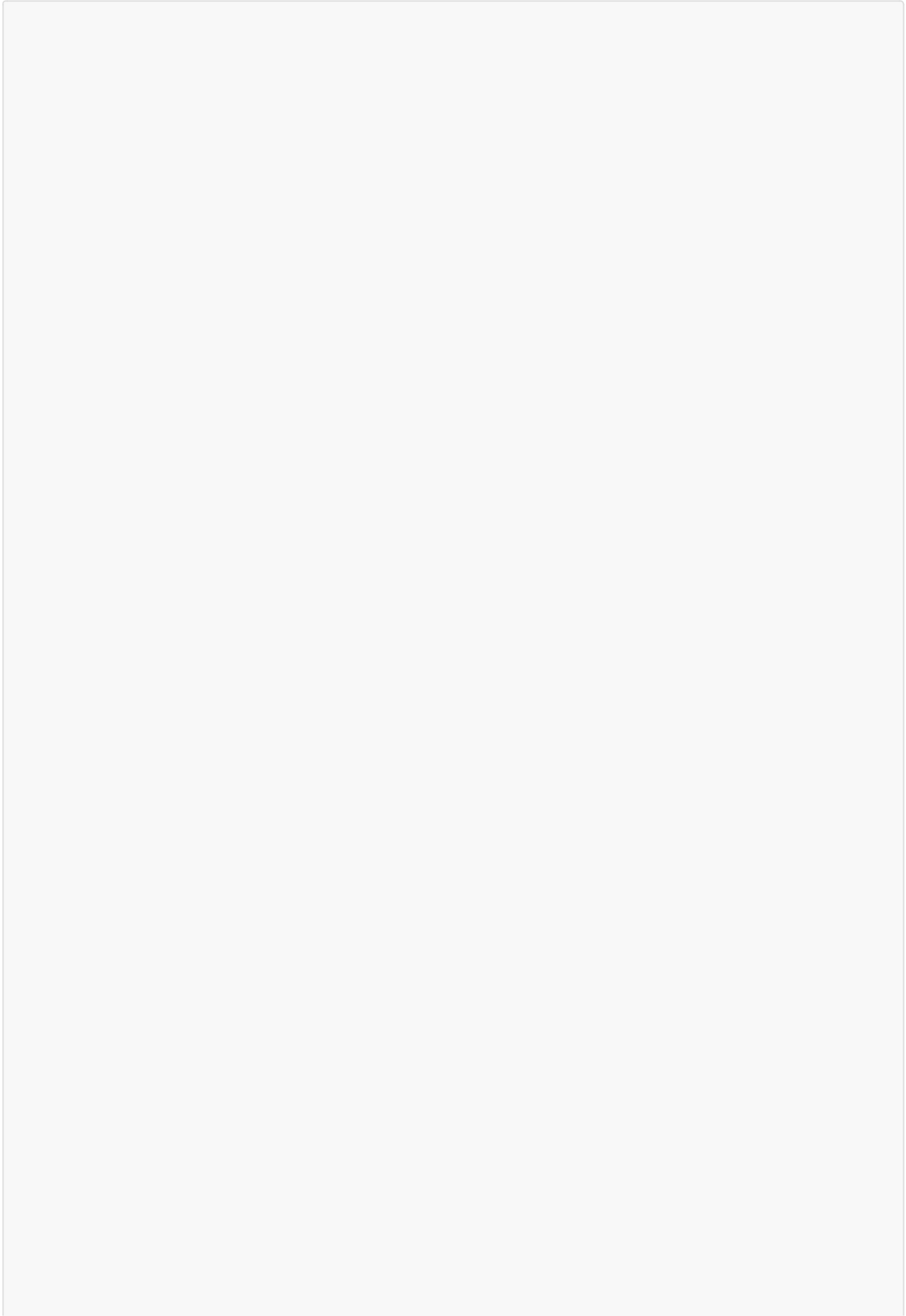
data: [1, 2, 3],

maxNum: 3

}

}

下面对List.js进行改造：

import React from 'react';

import { connect } from 'dva';

const namespace = 'list';

const mapStateToProps = (state) => {

const listData = state[namespace].data;

return {

listData

};

};

@connect(mapStateToProps)

class List extends React.Component{

render(){

return (

<div>

<ul>

{

* 遍历值

this.props.listData.map((value,index) => { return <li key={index}>{value}</li>

})

}

</ul>

<button

onClick={()=>{ //为按钮添加点击事件

* let maxNum = this.state.maxNum + 1;
* let list = [...this.state.dataList, maxNum];
* this.setState({ //更新状态值
* dataList : list,
* maxNum : maxNum
* });

}}>

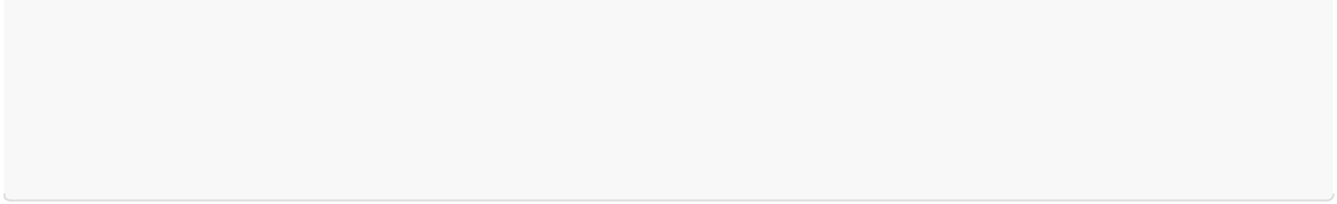
添加

</button>

</div>

);

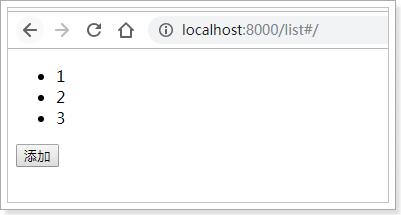
}



}

export default List;

测试：



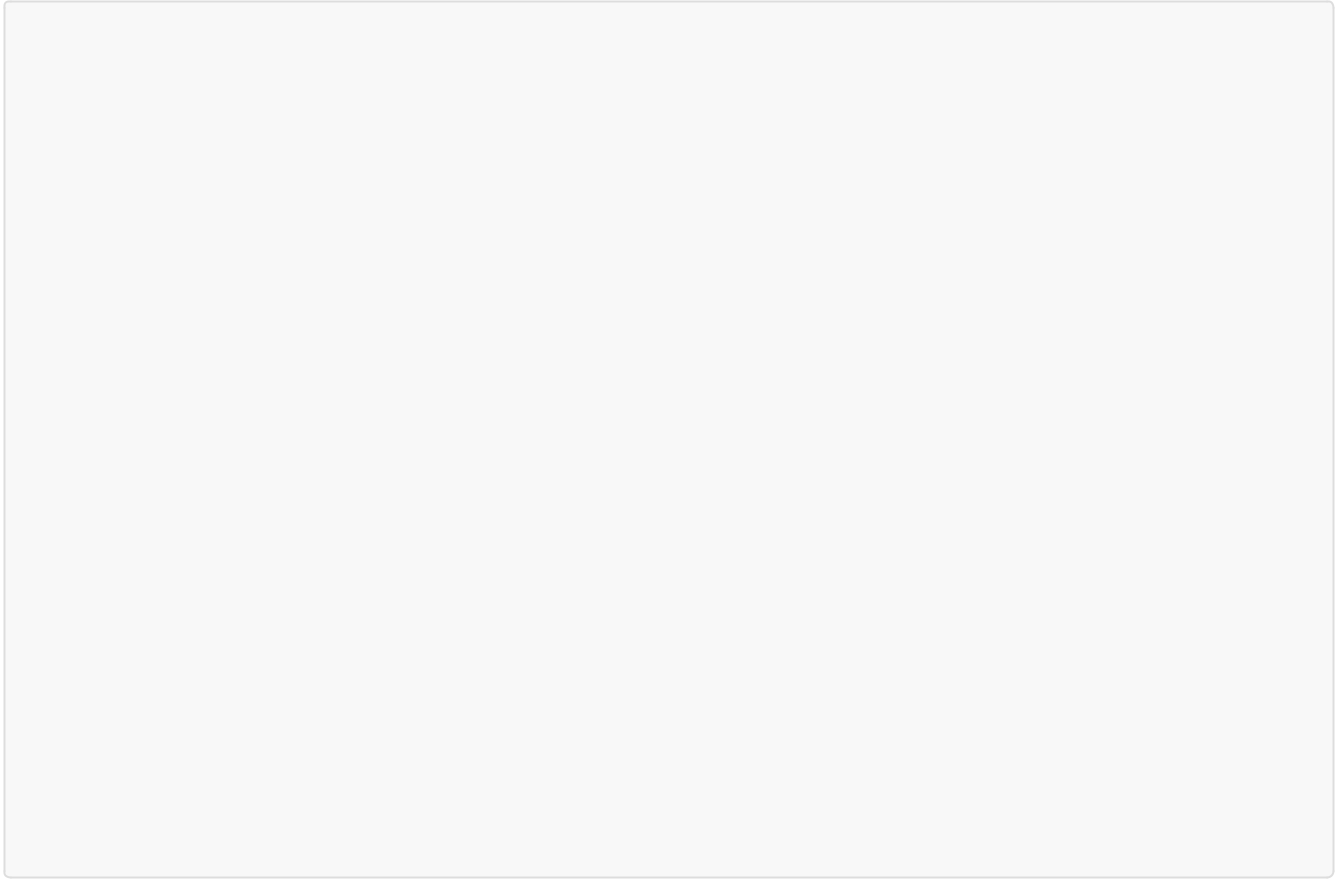
可以看到，效果是一样的。

流程说明：

1. umi框架启动，会自动读取models目录下model文件，即ListData.js中的数据
2. @connect修饰符的第一个参数，接收一个方法，该方法必须返回{}，将接收到model数据
3. 在全局的数据中，会有很多，所以需要通过namespace进行区分，所以通过state[namespace]进行获取数据
4. 拿到model数据中的data，也就是[1, 2, 3]数据，进行包裹{}后返回
5. 返回的数据，将被封装到this.props中，所以通过this.props.listData即可获取到model中的数据

刚刚只是将数据展现出来，如果点击按钮，需要修改state的值，怎么操作呢？

首先，在model中新增reducers方法，用于更新state中的数据：

export default {

namespace: 'list',

state: {

data: [1, 2, 3],

maxNum: 3

},

reducers : {

addNewData(state){ //state是更新前的对象

let maxNum = state.maxNum + 1;

let list = [...state.data, maxNum];

return { // 返回更新后的state对象

data : list,

maxNum : maxNum

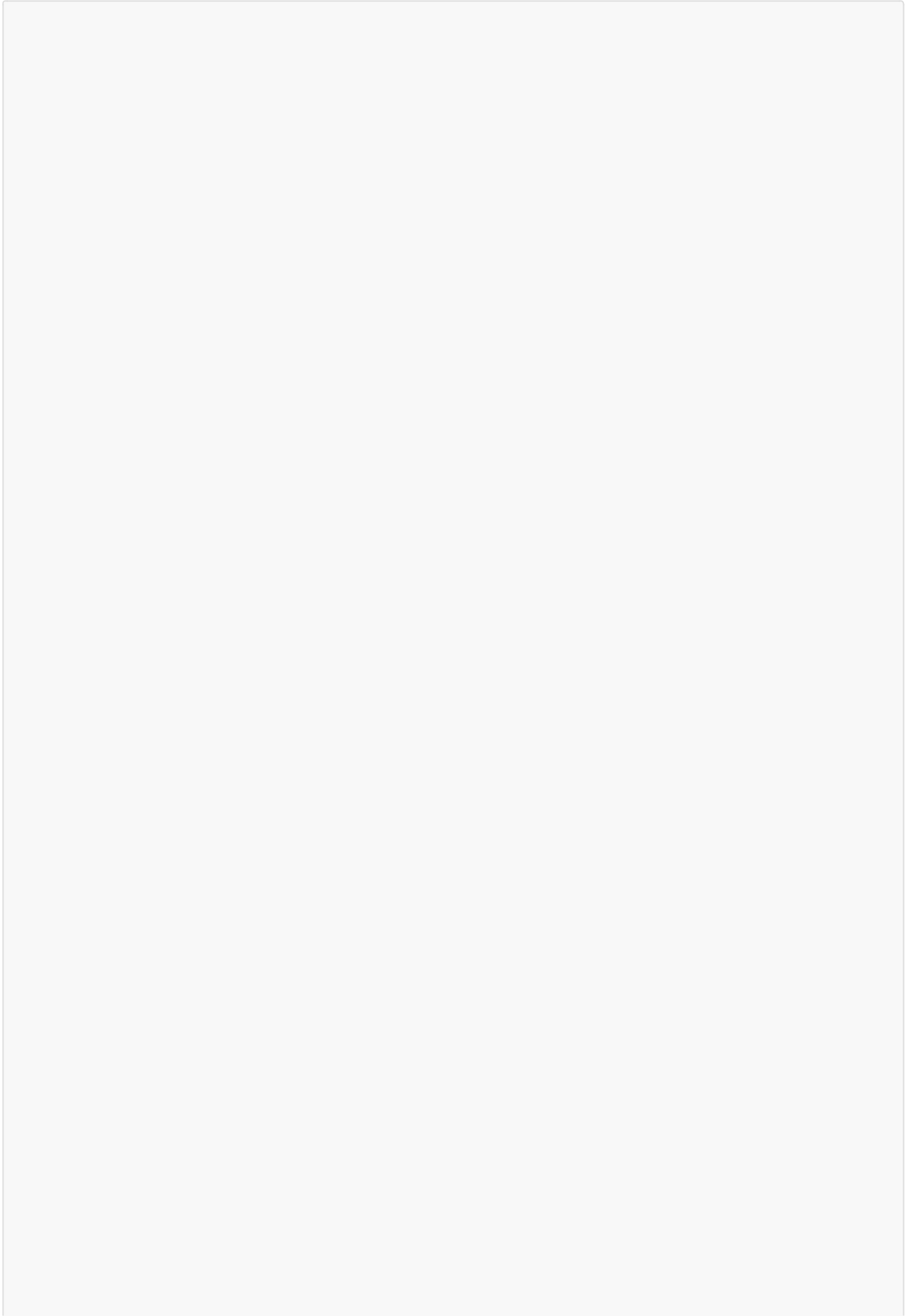
}

}

}

}

接下来修改List.js新增点击事件：

import React from 'react';

import { connect } from 'dva';

const namespace = 'list';

const mapStateToProps = (state) => {

const listData = state[namespace].data;

const maxNum = state[namespace].maxNum;

return {

listData, maxNum

};

};

const mapDispatchToProps = (dispatch) => { // 定义方法，dispatch是内置函数 return { //返回的这个对象将绑定到this.props对象中

addNewData : () =>{ // 定义方法

dispatch({ // 通过调用dispatch()方法，调用model中reducers的方法 type: namespace + "/addNewData" // 指定方法，格式：

namespace/方法名

});

}

}

}

@connect(mapStateToProps, mapDispatchToProps) //mapDispatchToProps：函数，将方法映射到props中

class List extends React.Component{

render(){

return (

<div>

<ul>

{

* 遍历值

this.props.listData.map((value,index) => { return <li key={index}>{value}</li>

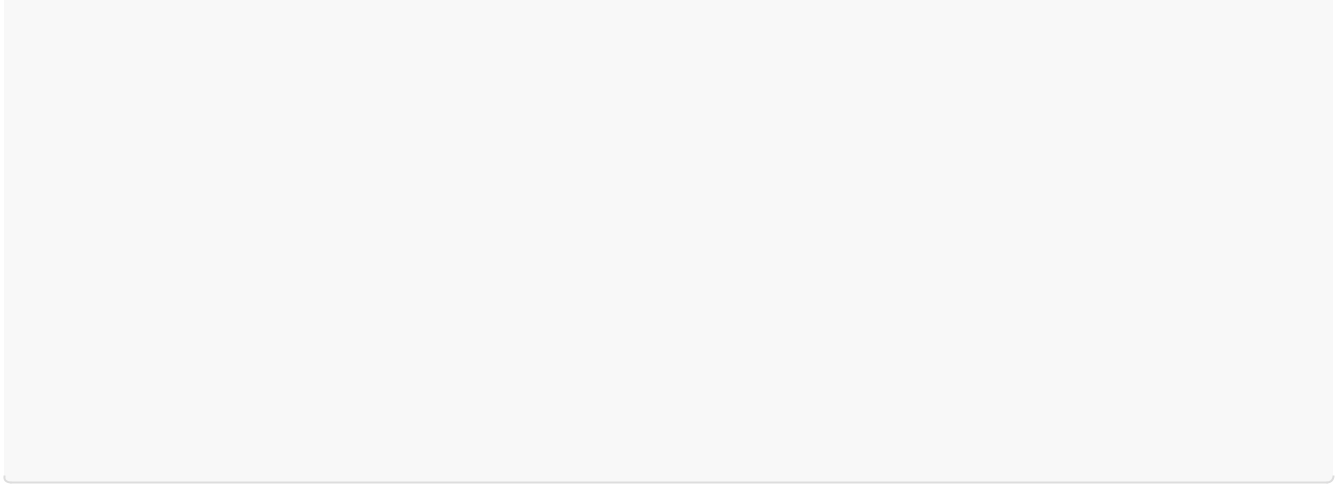
})

}

</ul>

<button

onClick={()=>{this.props.addNewData()}}> 添加



</button>

</div>

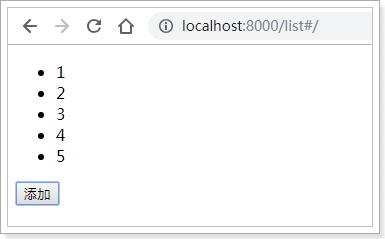
);

}

}

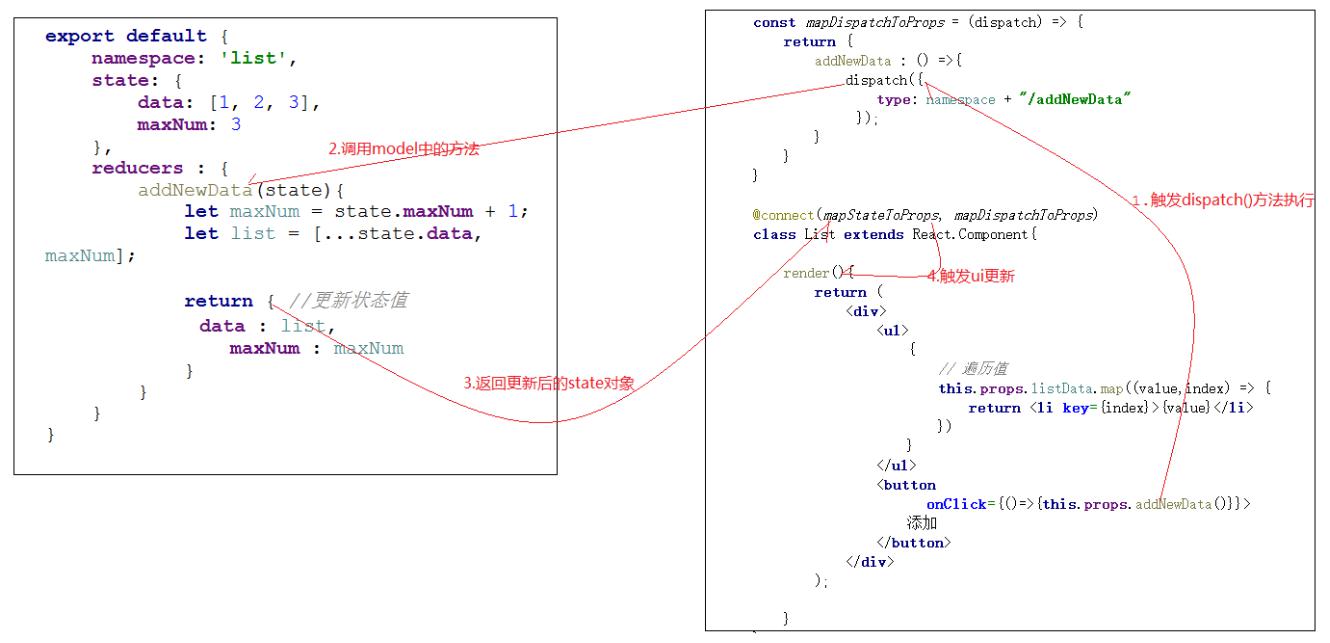
export default List;

测试：



测试结果，和之前实现效果一样。

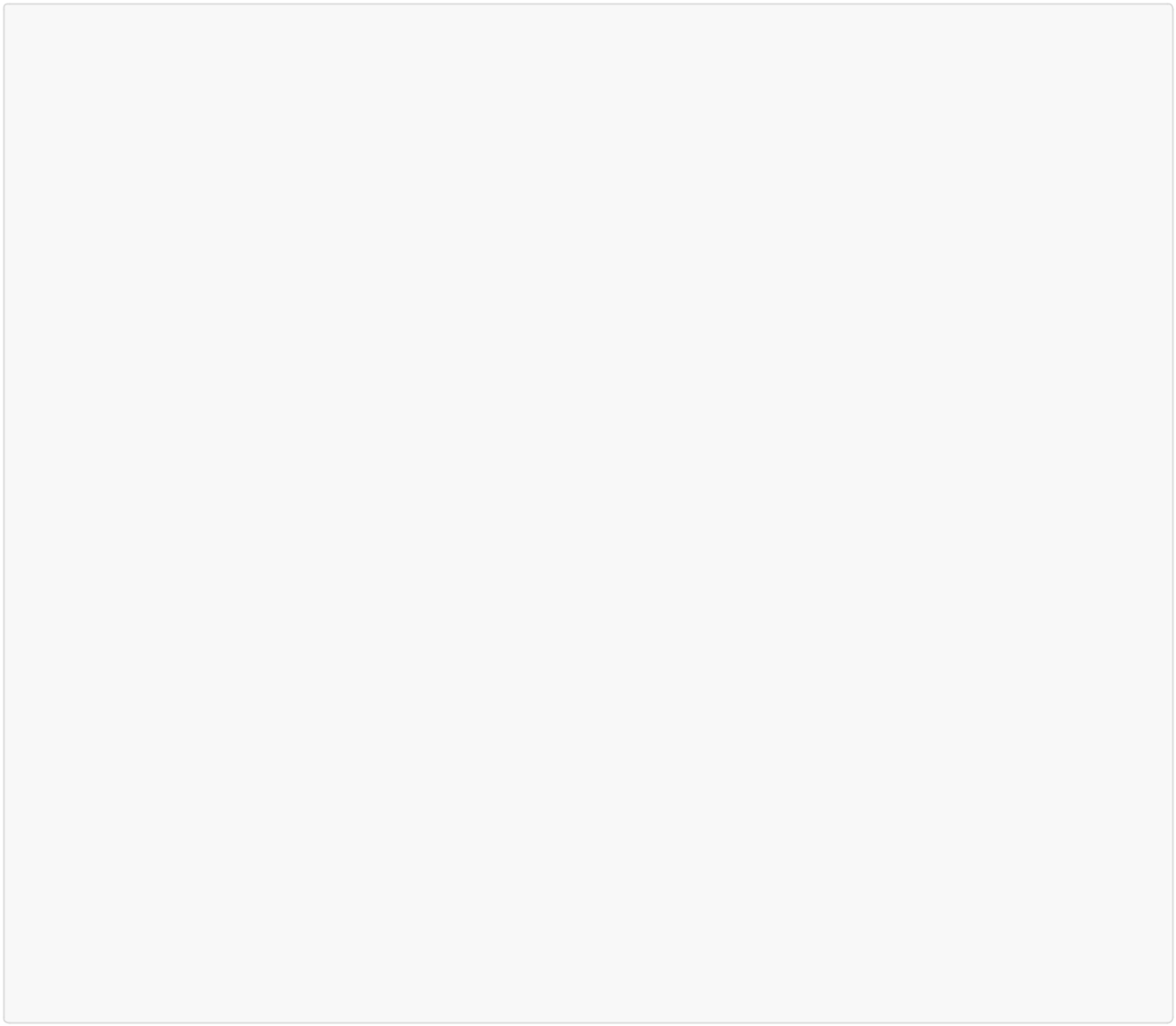
流程梳理如下：



1.1.3、在model中请求数据

前面我的数据是写死在model中的，实际开发中，更多的是需要异步加载数据，那么在model中如何异步加载数据呢？

首先，创建src下创建util目录，并且创建request.js文件，输入如下内容：（用于异步请求数据）



// import fetch from 'dva/fetch';

function checkStatus(response) {

if (response.status >= 200 && response.status < 300) { return response;

}

const error = new Error(response.statusText); error.response = response;

throw error;

}

/\*\*

* Requests a URL, returning a promise.

|  |  |  |  |
| --- | --- | --- | --- |
| \* @param | {string} url | | The URL we want to request |
| \* @param | {object} | [options] | The options we want to pass to "fetch" |
| \* @return {object} | |  | An object containing either "data" or "err" |

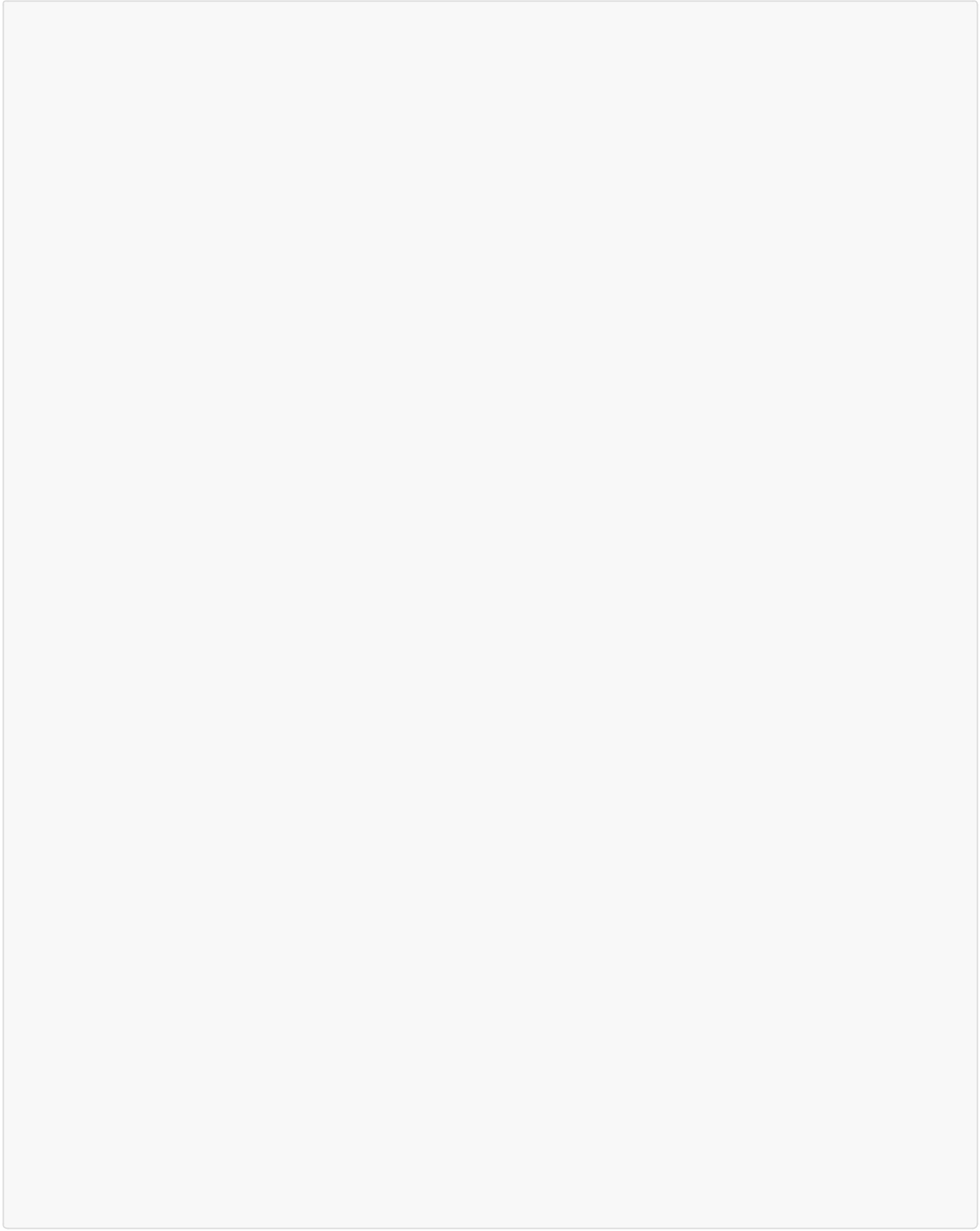
\*/

export default async function request(url, options) { const response = await fetch(url, options); checkStatus(response);

return await response.json();

}

然后，在model中新增请求方法：

import request from '../util/request';

export default {

namespace: 'list',

state: {

data: [],

maxNum: 0

},

reducers: {

addNewData(state, result) { //result就是拿到的结果数据

if(result.data){ //判断result中的data是否存在，如果存在，说明是初

始化数据，直接返回

return result.data;

}

let maxNum = state.maxNum + 1;

let list = [...state.data, maxNum];

return { //更新状态值

data: list,

maxNum: maxNum

}

}

},

effects: { //新增effects配置，用于异步加载数据

\*initData(params, sagaEffects) { //定义异步方法

const {call, put} = sagaEffects; //获取到call、put方法 const url = "/ds/list"; // 定义请求的url

let data = yield call(request, url); //执行请求 yield put({ // 调用reducers中的方法

type : "addNewData", //指定方法名

data : data //传递ajax回来的数据

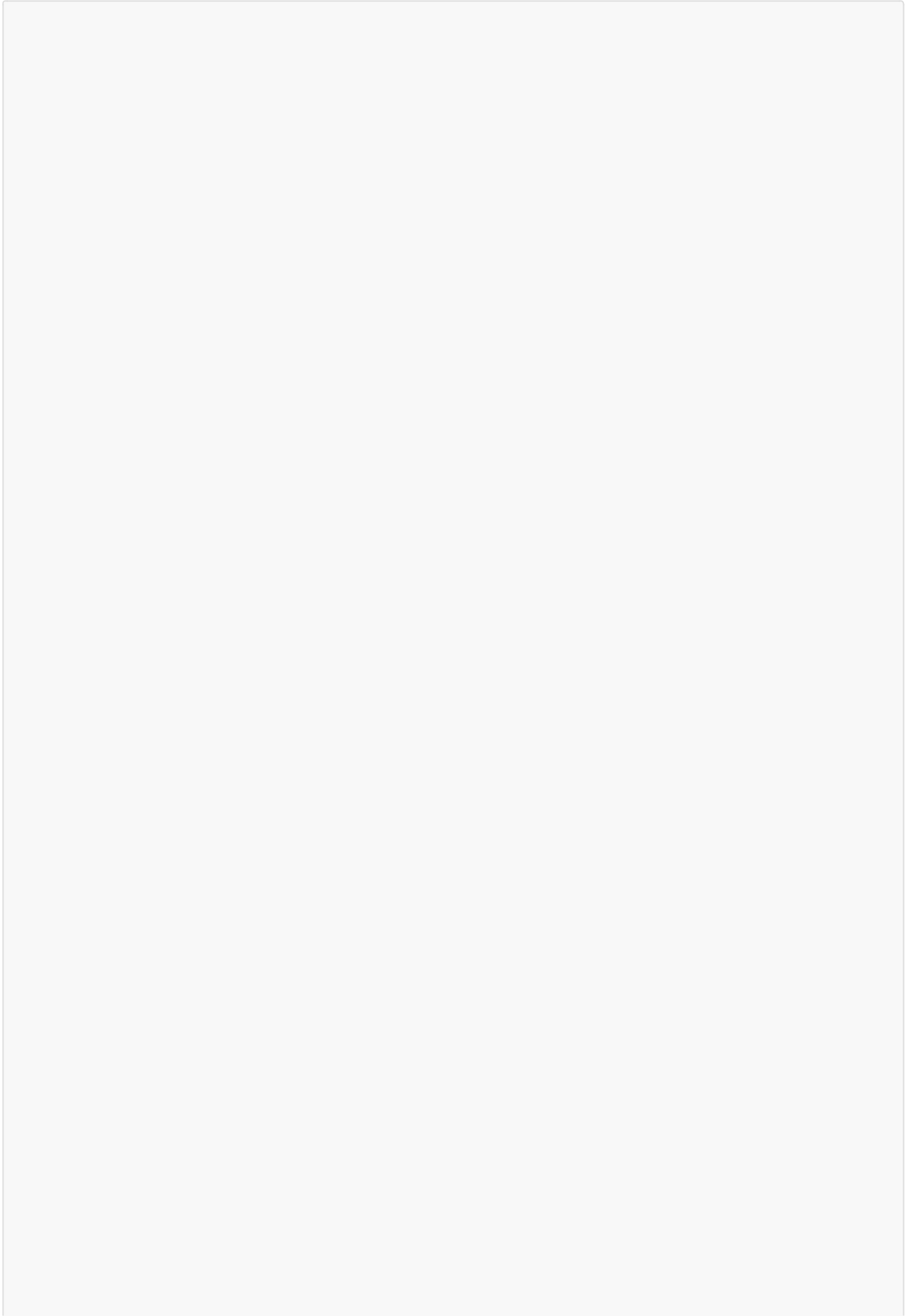
});

}

}

}

改造页面逻辑：

import React from 'react';

import { connect } from 'dva';

const namespace = 'list';

const mapStateToProps = (state) => {

const listData = state[namespace].data;

const maxNum = state[namespace].maxNum;

return {

listData, maxNum

};

};

const mapDispatchToProps = (dispatch) => { return {

addNewData : () =>{

dispatch({

type: namespace + "/addNewData"

});

},

initData : () => { //新增初始化方法的定义 dispatch({

type: namespace + "/initData"

});

}

}

}

@connect(mapStateToProps, mapDispatchToProps)

class List extends React.Component{

componentDidMount(){

this.props.initData(); //组件加载完后进行初始化操作

}

render(){

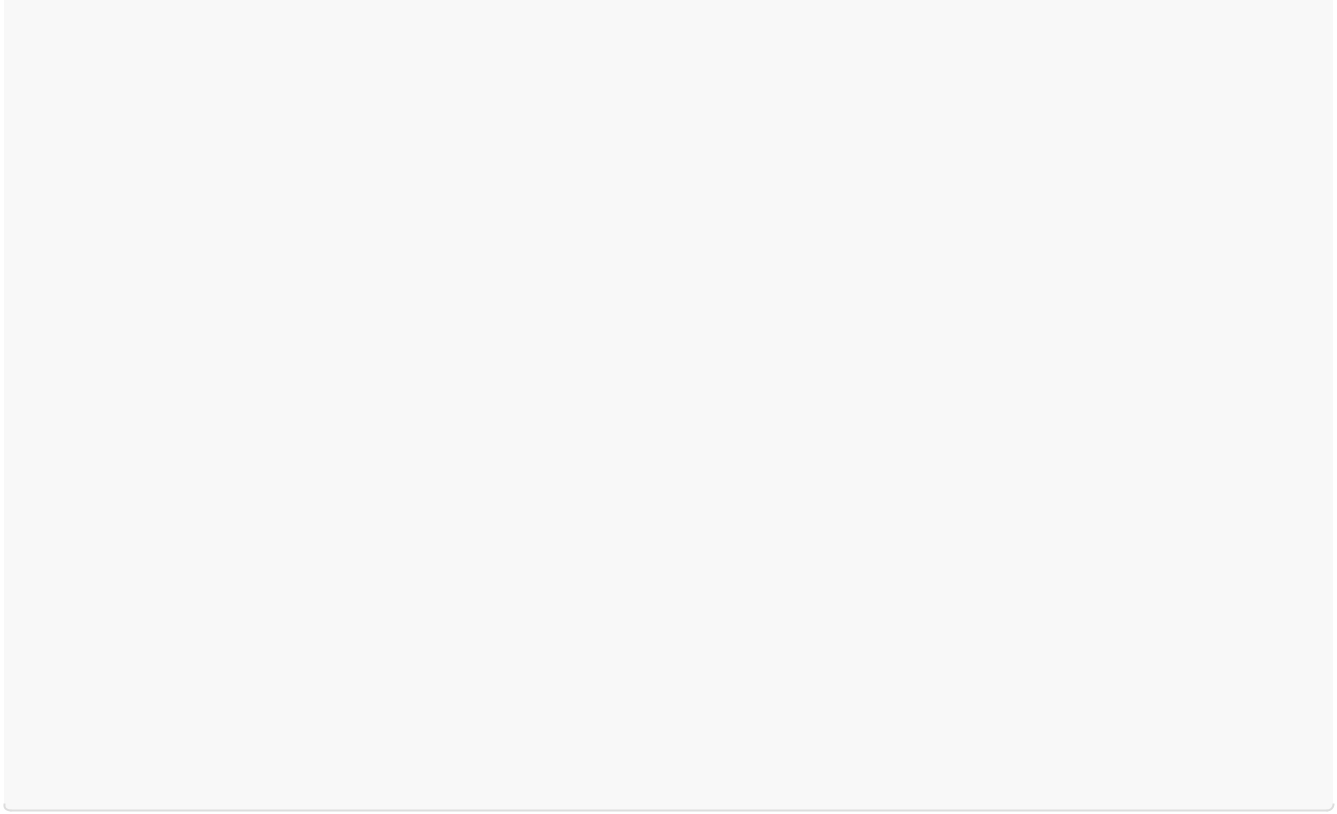
return (

<div>

<ul>

{

* 遍历值



this.props.listData.map((value,index) => { return <li key={index}>{value}</li>

})

}

</ul>

<button

onClick={()=>{this.props.addNewData()}}> 添加

</button>

</div>

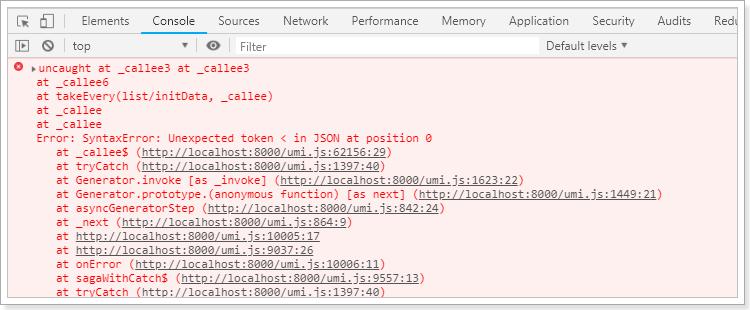
);

}

}

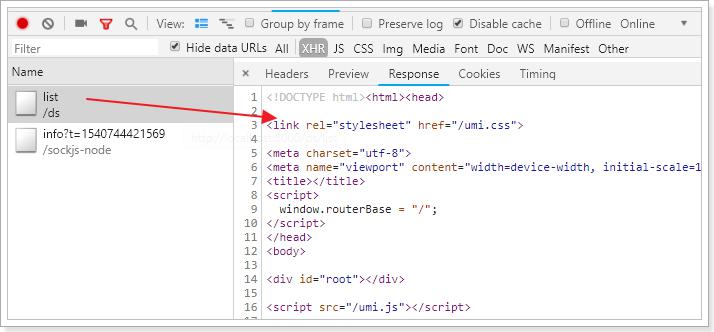
export default List;

测试：



测试结果，发现会报错，原因是返回的数据不是json导致，解析出错。

查看下请求：

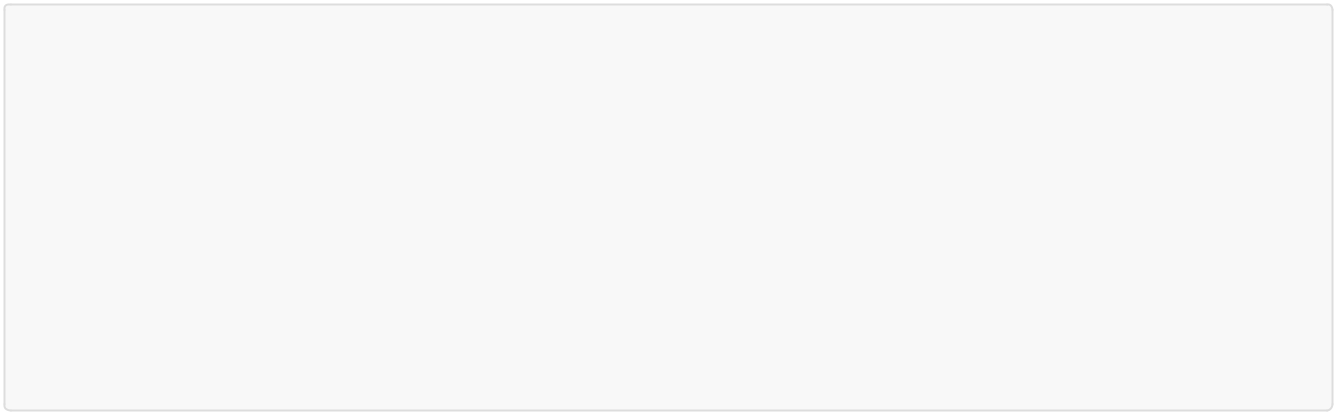


可以看到，返回的是html代码，所以会导致出错。

1.1.4、mock数据

umi中支持对请求的模拟，由于我们现在没有真正的服务可以返回数据，所以才需要模拟。

在项目根目录下创建mock目录，然后创建MockListData.js文件，并且输入如下内容：



export default {

'get /ds/list': function (req, res) { //模拟请求返回数据 res.json({

data: [1, 2, 3, 4],

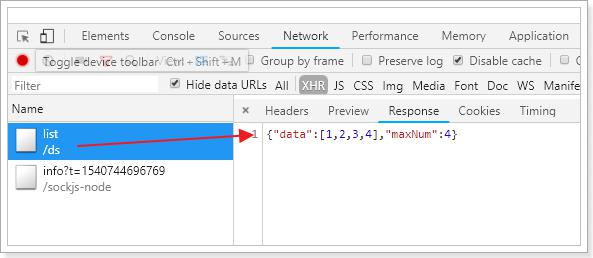
maxNum: 4

});

}

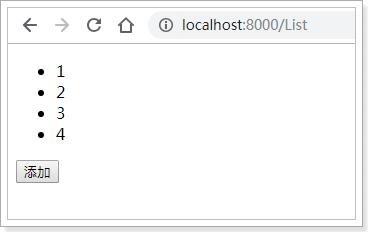
}

进行测试：



发现，可以正常返回数据了。

页面效果也正常了:



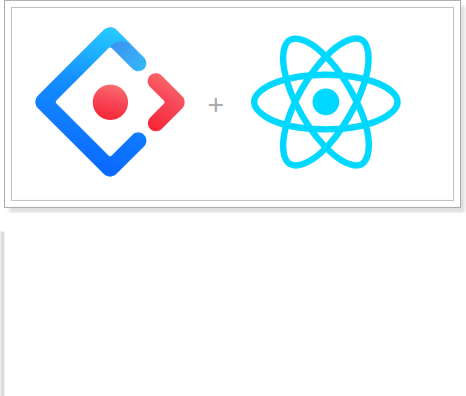
2、Ant Design入门



2.1、什么是Ant Design？

Ant Design是阿里蚂蚁金服团队基于React开发的ui组件，主要用于中后台系统的使用。

官网：<https://ant.design/index-cn>



设计语言：

随着商业化的趋势，越来越多的企业级产品对更好的用户体验有了进一步的要求。带着这样的一个终极目标，我们（蚂蚁金服体验技术部）经过大量的项目实践和总结，逐步打磨出一个服务于企业级产品的设计体系 Ant Design。基于『确定』和『自然』的设计价值观，通过模块化的解决方案，降低冗余的生产成本，让设计者专注于更好的用户体验。

特性：

 提炼自企业级中后台产品的交互语言和视觉风格。

 开箱即用的高质量 React 组件。

 使用 TypeScript 构建，提供完整的类型定义文件。

 全链路开发和设计工具体系。

2.2、开始使用

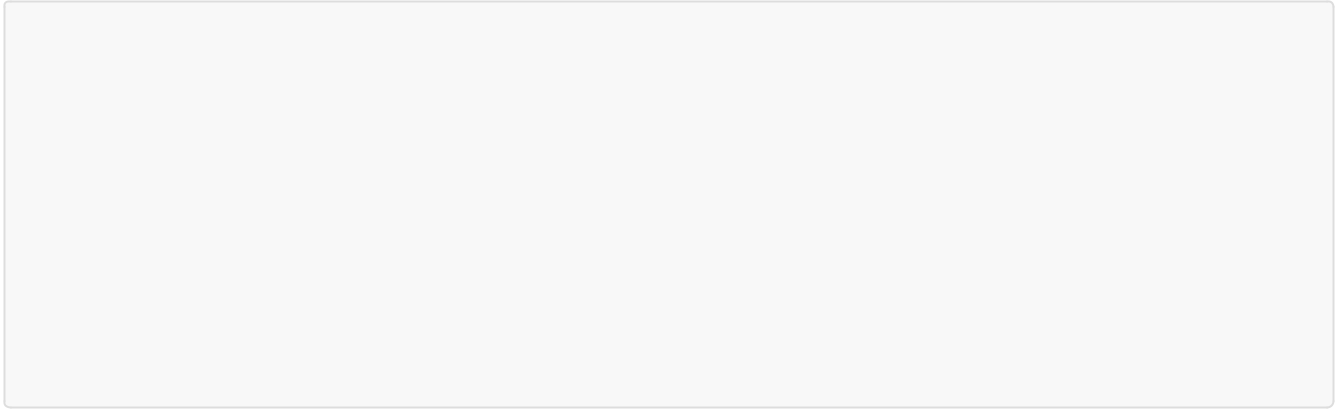
2.2.1、引入Ant Design

Ant Design 是一个服务于企业级产品的设计体系，组件库是它的 React 实现，antd 被发布为一个 npm 包方便开发者安装并使用。

* umi 中，你可以通过在插件集 umi-plugin-react 中配置 antd 打开 antd 插件，antd

插件会帮你引入 antd 并实现按需编译。

在config.js文件中进行配置：

export default {

plugins: [

['umi‐plugin‐react', {

dva: true, // 开启dva功能

antd: true // 开启Ant Design功能

}]

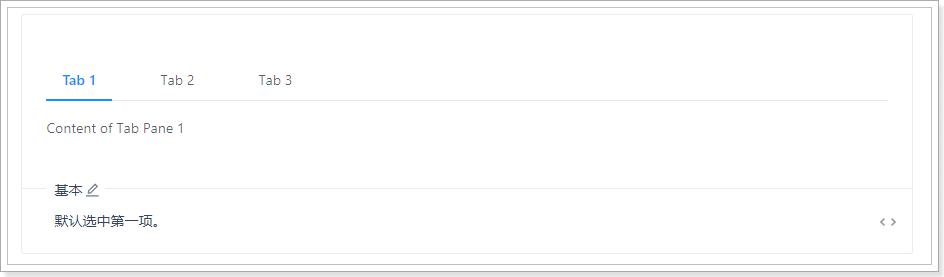
]

};

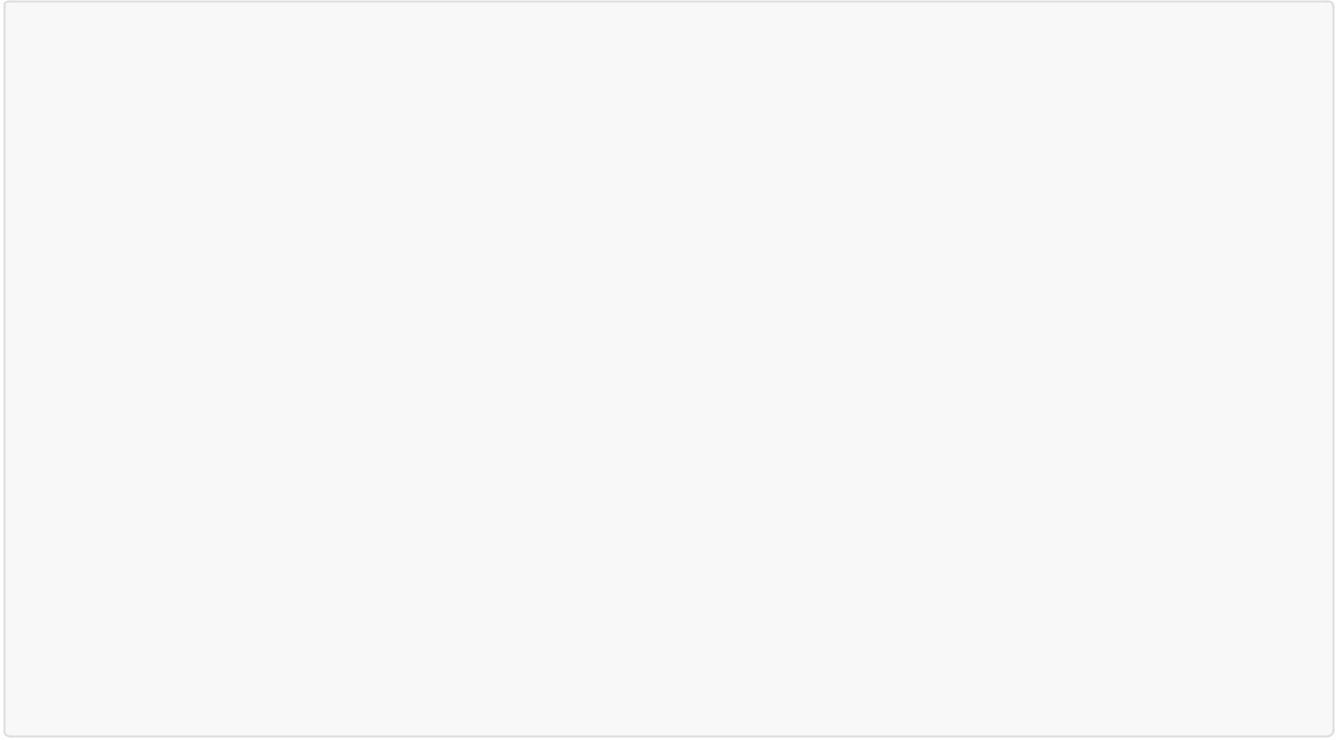
2.2.2、小试牛刀

接下来，我们开始使用antd的组件，以tabs组件为例，地址：<https://ant.design/components/tabs-cn/>

效果：



看下官方给出的使用示例：

import { Tabs } from 'antd';

const TabPane = Tabs.TabPane;

function callback(key) {

console.log(key);

}

ReactDOM.render(

<Tabs defaultActiveKey="1" onChange={callback}>

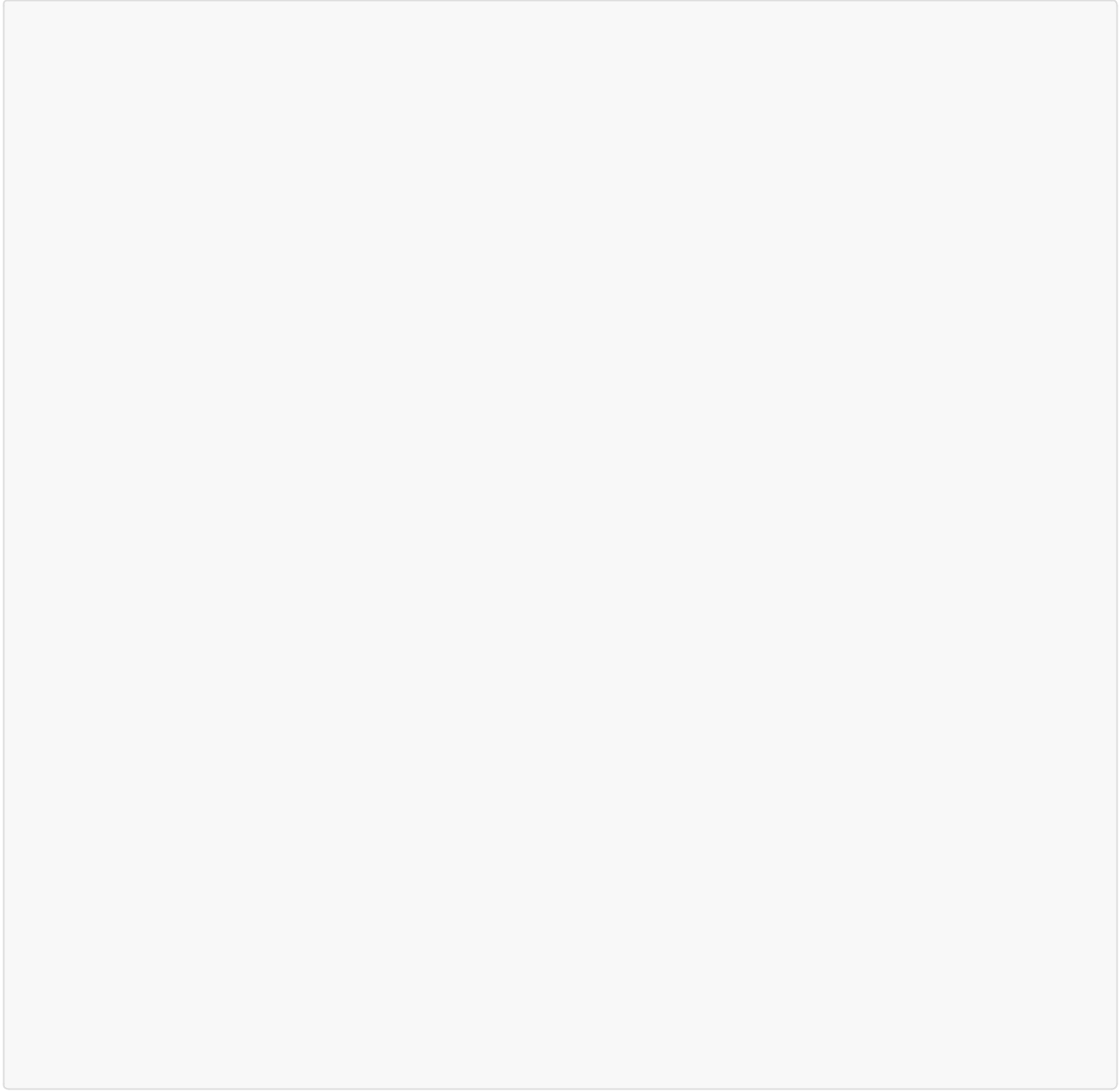
<TabPane tab="Tab 1" key="1">Content of Tab Pane 1</TabPane> <TabPane tab="Tab 2" key="2">Content of Tab Pane 2</TabPane> <TabPane tab="Tab 3" key="3">Content of Tab Pane 3</TabPane>

</Tabs>,

mountNode);

下面我们参考官方给出的示例，进行使用：

创建MyTabs.js文件：

import React from 'react'

import {Tabs} from 'antd'

const TabPane = Tabs.TabPane;

const callback = (key) => {

console.log(key);

}

class MyTabs extends React.Component {

render() {

return (

<Tabs defaultActiveKey="1" onChange={callback}>

<TabPane tab="Tab 1" key="1">Content of Tab Pane

1</TabPane>

<TabPane tab="Tab 2" key="2">Content of Tab Pane

2</TabPane>

<TabPane tab="Tab 3" key="3">Content of Tab Pane

3</TabPane>

</Tabs>

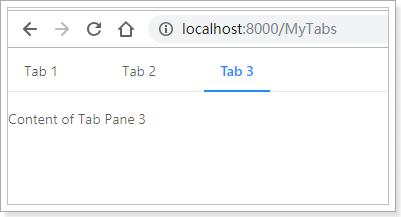
)

}

}

export default MyTabs;

效果：

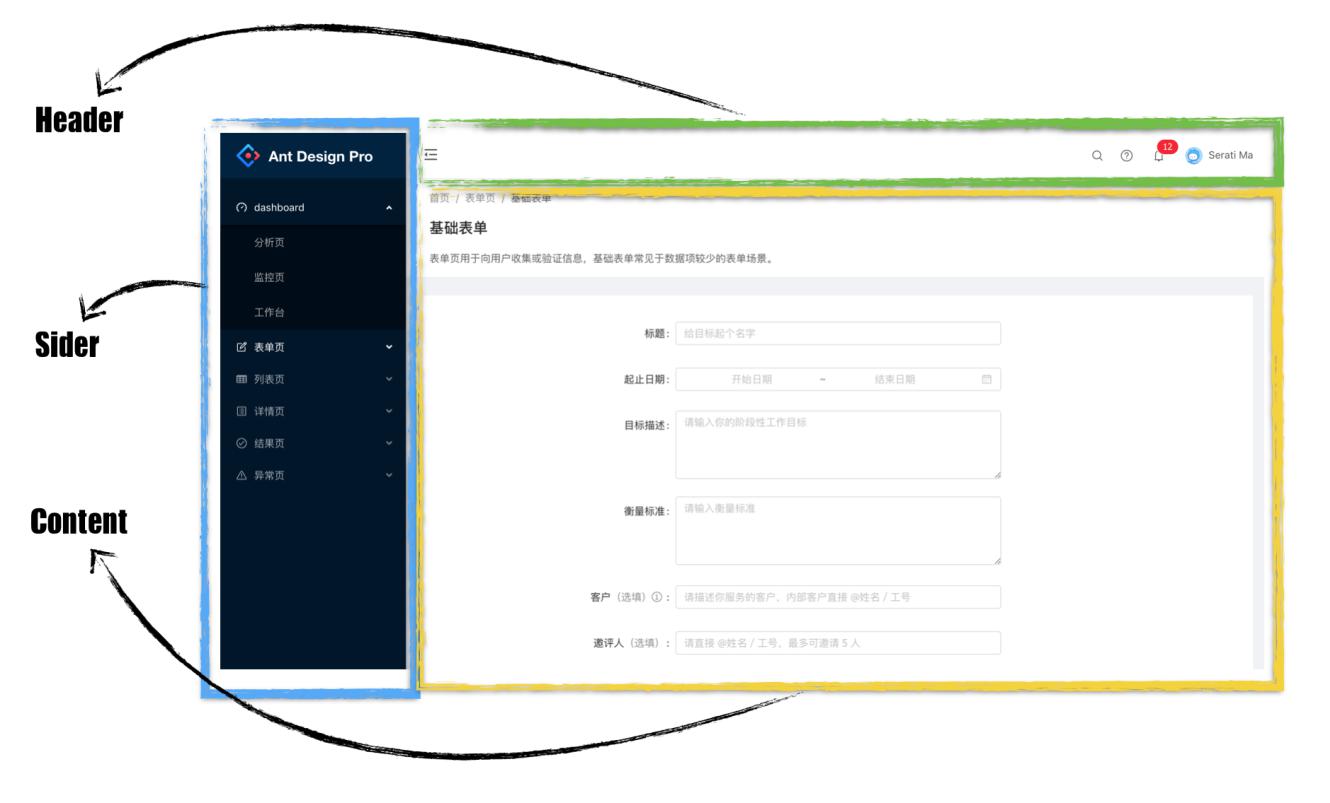


到此，我们已经掌握了antd组件的基本使用。

2.3、布局

antd布局：<https://ant.design/components/layout-cn/>

在后台系统页面布局中，往往是经典的三部分布局，像这样：



下面，我们通过antd组件来完成这个布局。

2.3.1、组件概述



 Layout ：布局容器，其下可嵌套 Header Sider Content Footer 或 Layout 本

身，可以放在任何父容器中。



 Header ：顶部布局，自带默认样式，其下可嵌套任何元素，只能放在 Layout 中。  Sider ：侧边栏，自带默认样式及基本功能，其下可嵌套任何元素，只能放在



Layout 中。



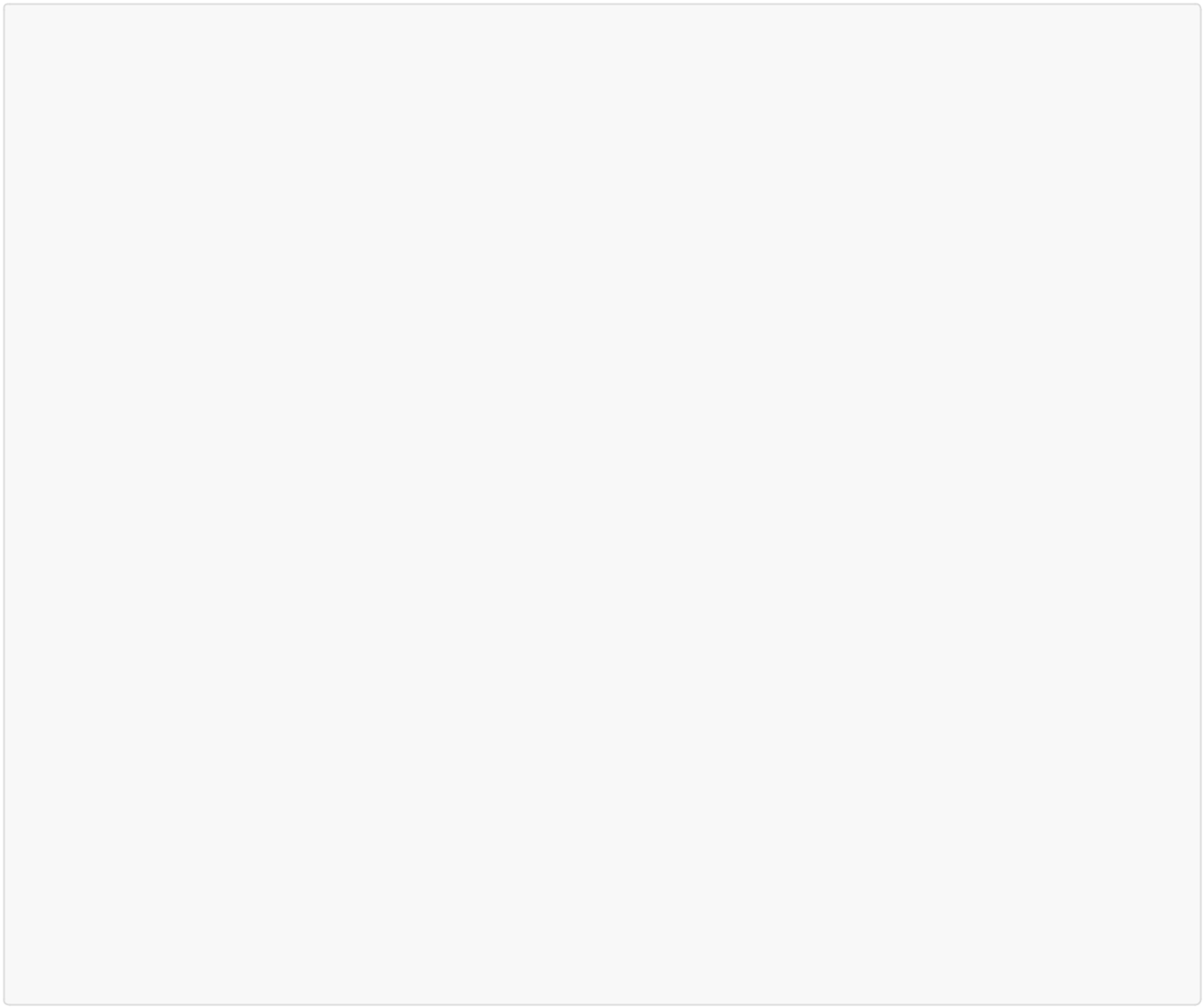
 Content ：内容部分，自带默认样式，其下可嵌套任何元素，只能放在 Layout 中。



 Footer ：底部布局，自带默认样式，其下可嵌套任何元素，只能放在 Layout 中。

2.3.2、搭建整体框架

在src目录下创建layouts目录，并且在layouts目录下创建index.js文件，写入内容：



import React from 'react'

import { Layout } from 'antd';

const { Header, Footer, Sider, Content } = Layout;

class BasicLayout extends React.Component{

render(){

return (

<Layout>

<Sider>Sider</Sider>

<Layout>

<Header>Header</Header>

<Content>Content</Content>

<Footer>Footer</Footer>

</Layout>

</Layout>

);

}

}

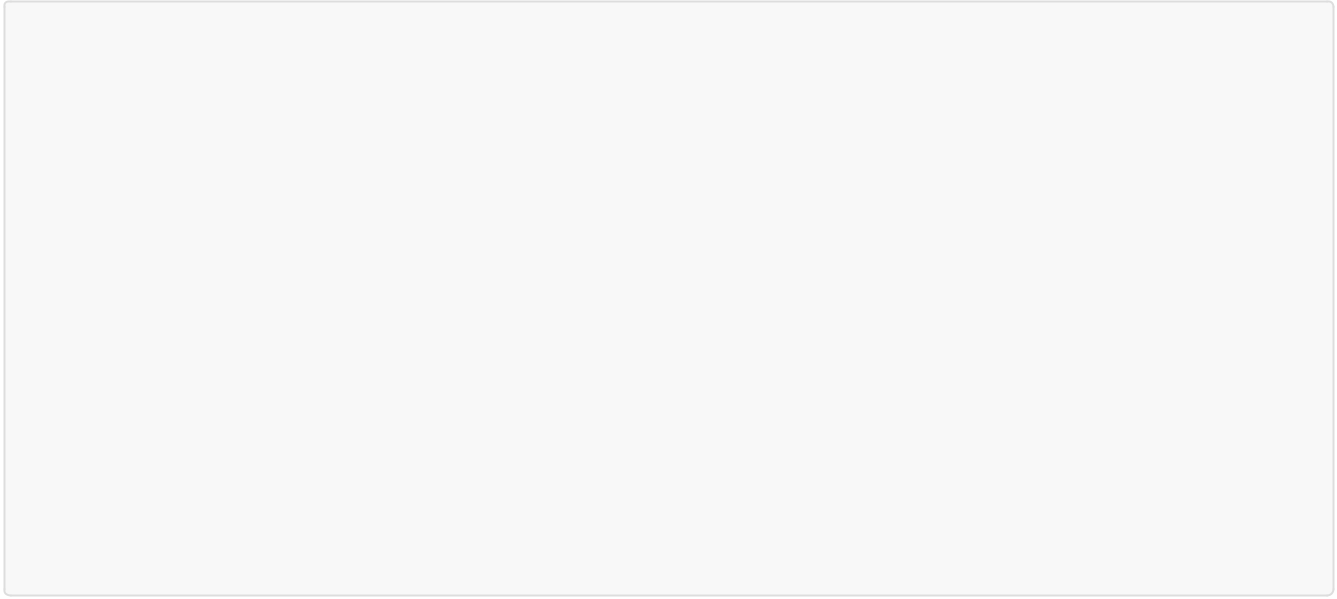
export default BasicLayout;



需要特别说明的是，在umi中约定的目录结构中，layouts/index.js文件将被作为全局的布局文件。

接下来，配置路由：（非必须）

config.js文件：

export default {

plugins: [

['umi‐plugin‐react', {

dva: true, // 开启dva功能

antd: true // 开启Ant Design功能

}]

],

routes: [{

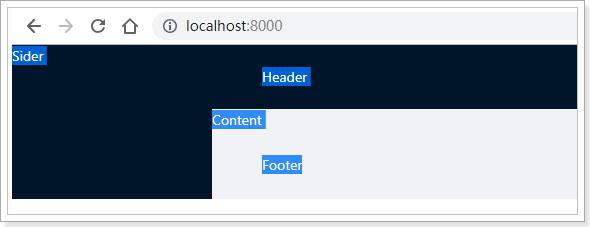
path: '/',

component: '../layouts' //配置布局路由

}]

};

进行页面访问：

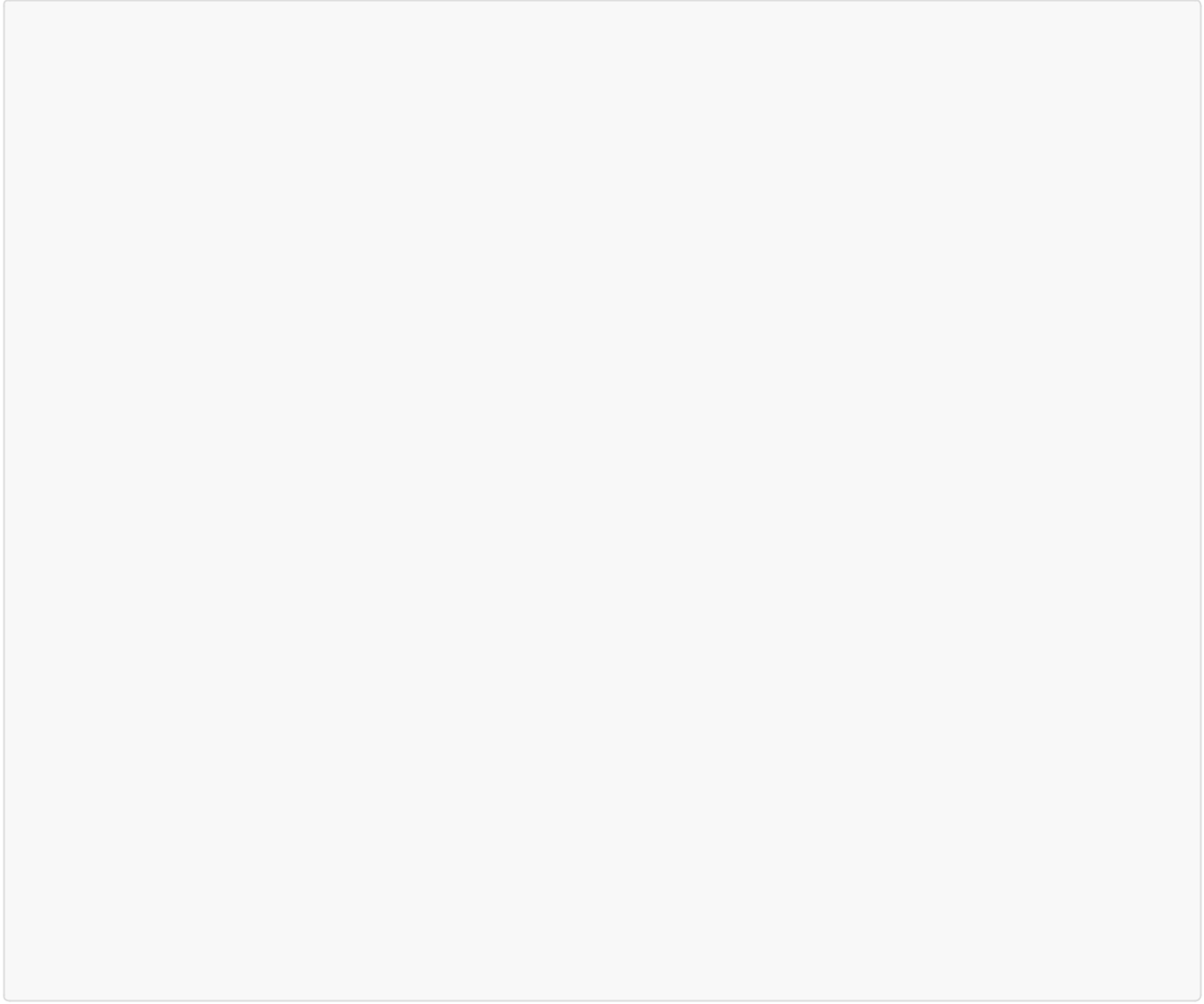


可以看到，布局已经生成，只是样式优点丑。

2.3.3、子页面使用布局

前面所定义的布局是全局布局，那么，在子页面中如何使用这个全局布局呢？

首先，需要在布局文件中，将Content内容替换成{this.props.children}，意思是引入传递的内容。

import React from 'react'

import { Layout } from 'antd';

const { Header, Footer, Sider, Content } = Layout;

class BasicLayout extends React.Component{

render(){

return (

<Layout>

<Sider>Sider</Sider>

<Layout>

<Header>Header</Header>

<Content>{this.props.children}</Content> <Footer>Footer</Footer>

</Layout>

</Layout>

);

}

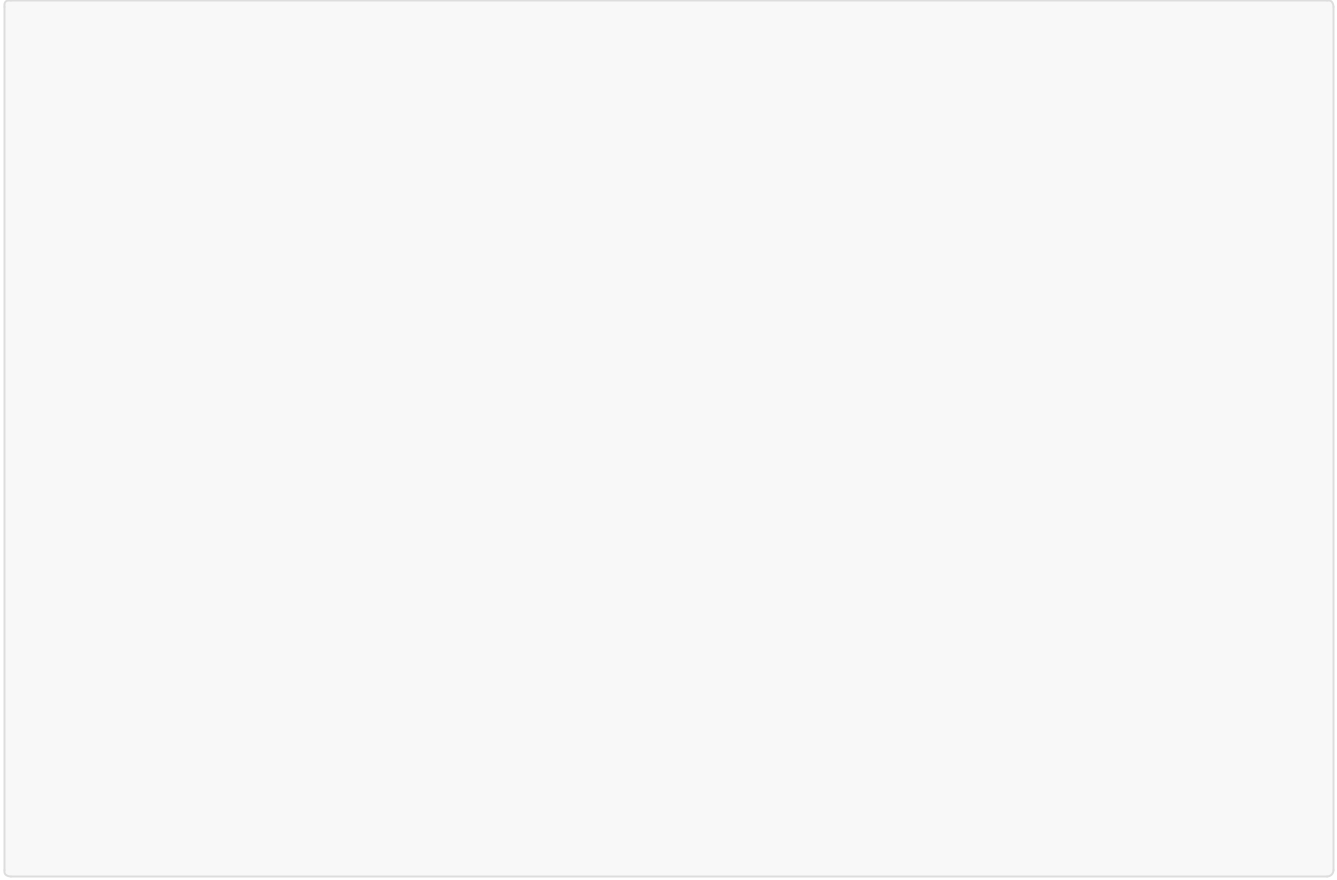
}

export default BasicLayout;

接下来配置路由（注意，在布局路由下面进行配置）：



说明：下面的路由配置，是表明你需要通过手动配置的方式上进行访问页面，而不采用umi默认的路由方式。

export default {

plugins: [

['umi‐plugin‐react', {

dva: true, // 开启dva功能

antd: true // 开启Ant Design功能

}]

],

routes: [{

path: '/',

component: '../layouts', //配置布局路由

routes: [ //在这里进行配置子页面

{

path: '/myTabs',

component: './myTabs'

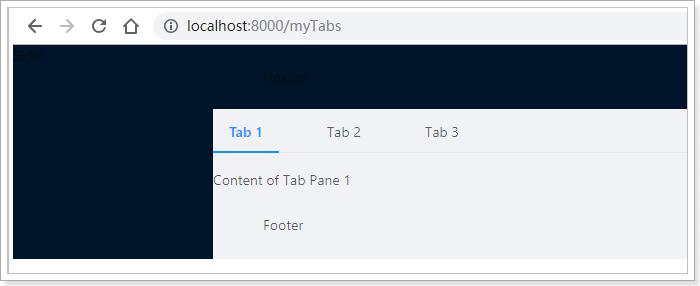
}

]

}]

};

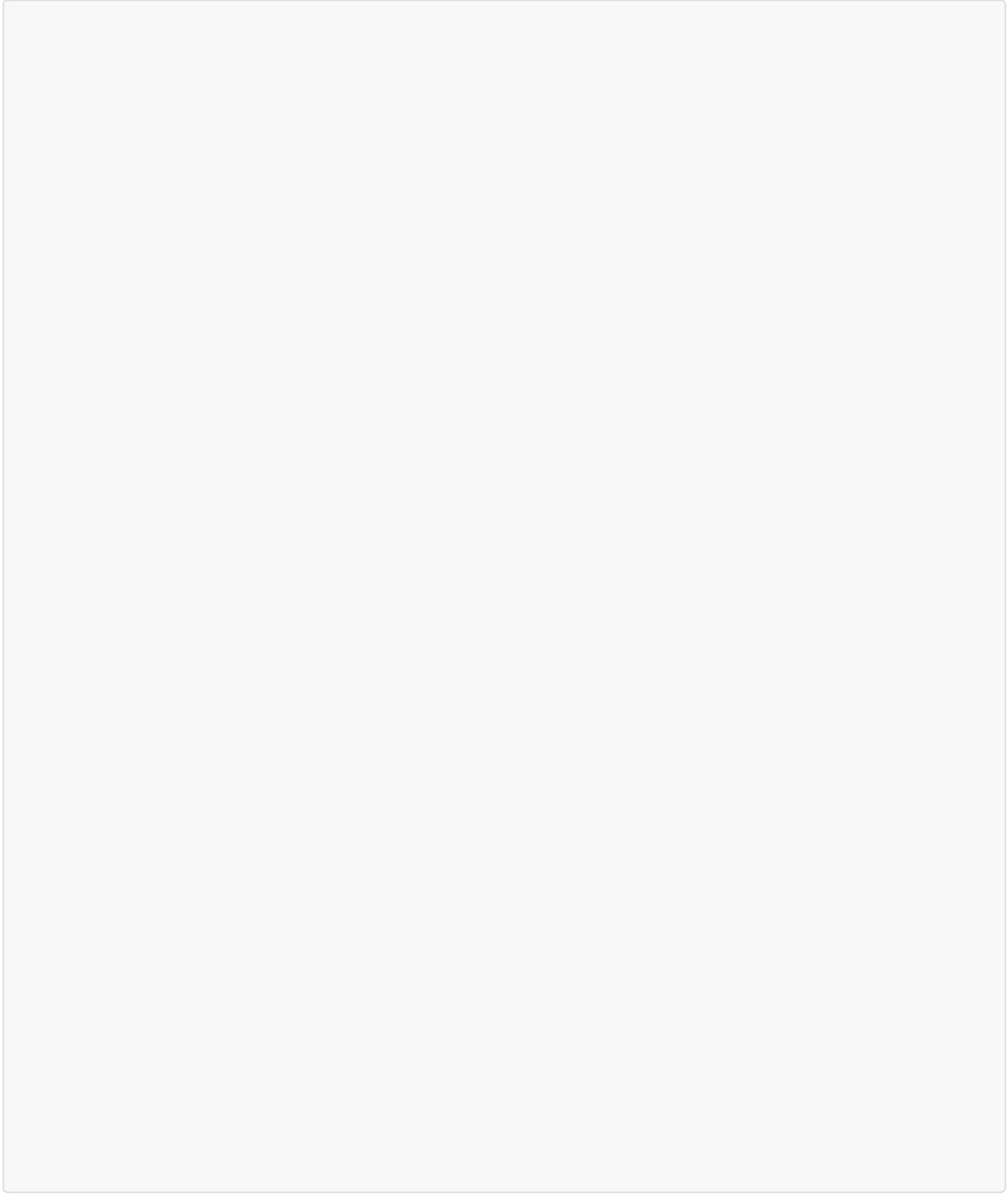
进行访问测试：



可以看到，在MyTabs组件中已经应用了全局的布局。其他子页面也就同理了。

2.3.4、美化页面

接下来，对页面做一些美化的工作：

import React from 'react'

import { Layout } from 'antd';

const { Header, Footer, Sider, Content } = Layout;

class BasicLayout extends React.Component{

render(){

return (

<Layout>

<Sider width={256} style={{ minHeight: '100vh', color:

'white' }}>

Sider

</Sider>

<Layout>

<Header style={{ background: '#fff', textAlign:

'center', padding: 0 }}>Header</Header>

<Content style={{ margin: '24px 16px 0' }}>

<div style={{ padding: 24, background: '#fff',

minHeight: 360 }}>

{this.props.children}

</div>

</Content>

<Footer style={{ textAlign: 'center' }}>后台系统 ©2018 Created by 黑马程序员</Footer>

</Layout>

</Layout>

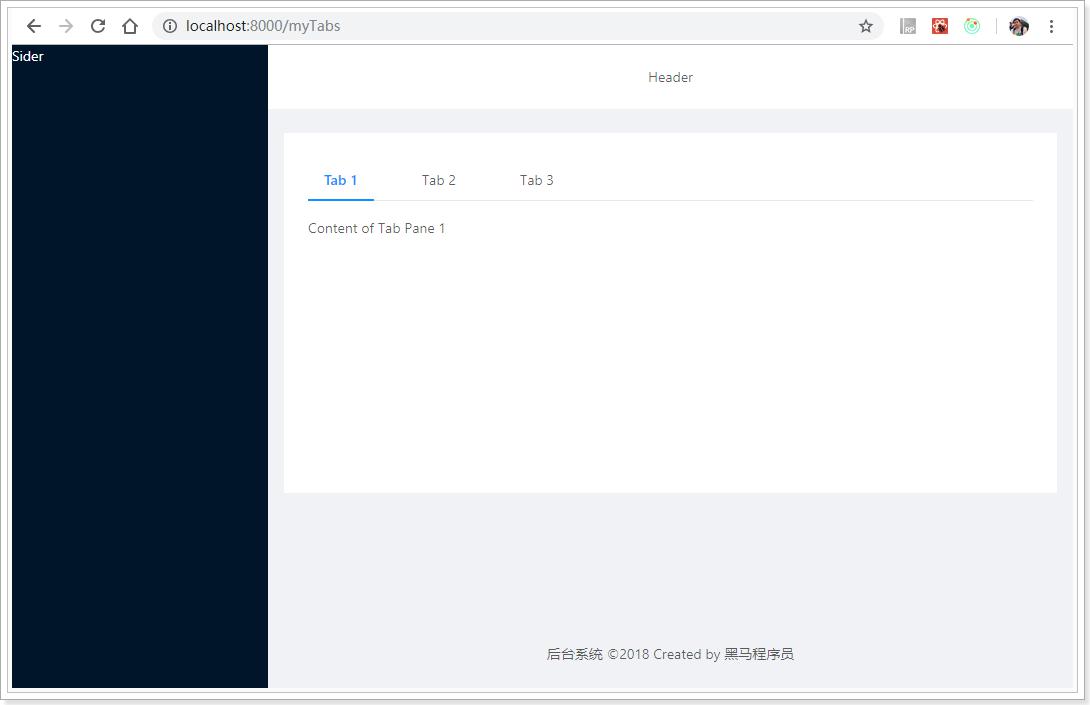
);

}

}

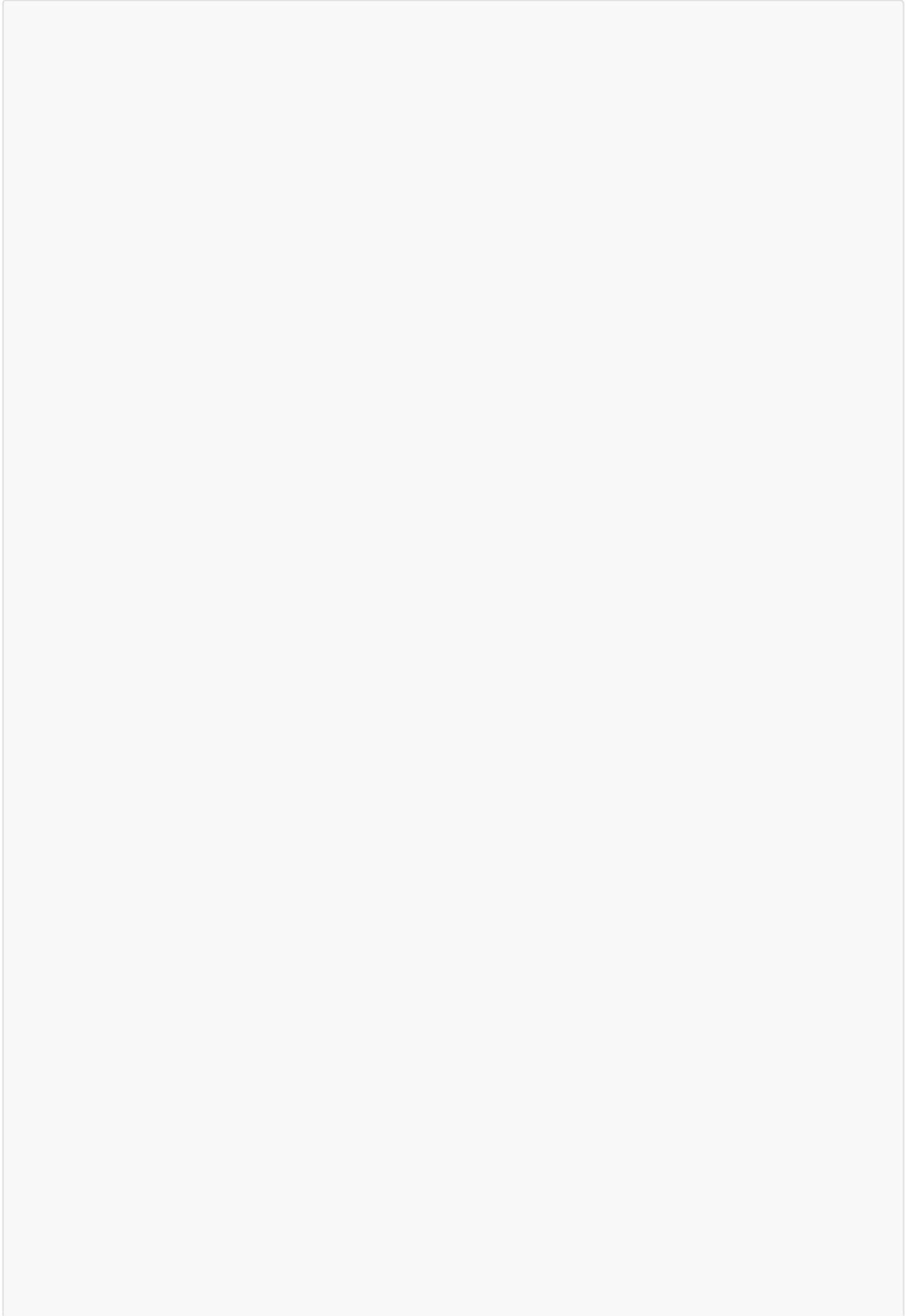
export default BasicLayout;

效果：



2.3.5、引入导航条

使用Menu组件作为导航条：<https://ant.design/components/menu-cn/>

import React from 'react'

import {Layout, Menu, Icon} from 'antd';

const {Header, Footer, Sider, Content} = Layout; const SubMenu = Menu.SubMenu;

class BasicLayout extends React.Component {

constructor(props){

super(props);

this.state = {

collapsed: false,

}

}

render() {

return (

<Layout>

<Sider width={256} style={{minHeight: '100vh', color:

'white'}}>

<div style={{ height: '32px', background:

'rgba(255,255,255,.2)', margin: '16px'}}/> <Menu

defaultSelectedKeys={['2']}

defaultOpenKeys={['sub1']}

mode="inline"

theme="dark"

inlineCollapsed={this.state.collapsed}

>

<Menu.Item key="1">

<Icon type="pie‐chart"/>

<span>Option 1</span>

</Menu.Item>

<Menu.Item key="2">

<Icon type="desktop"/>

<span>Option 2</span>

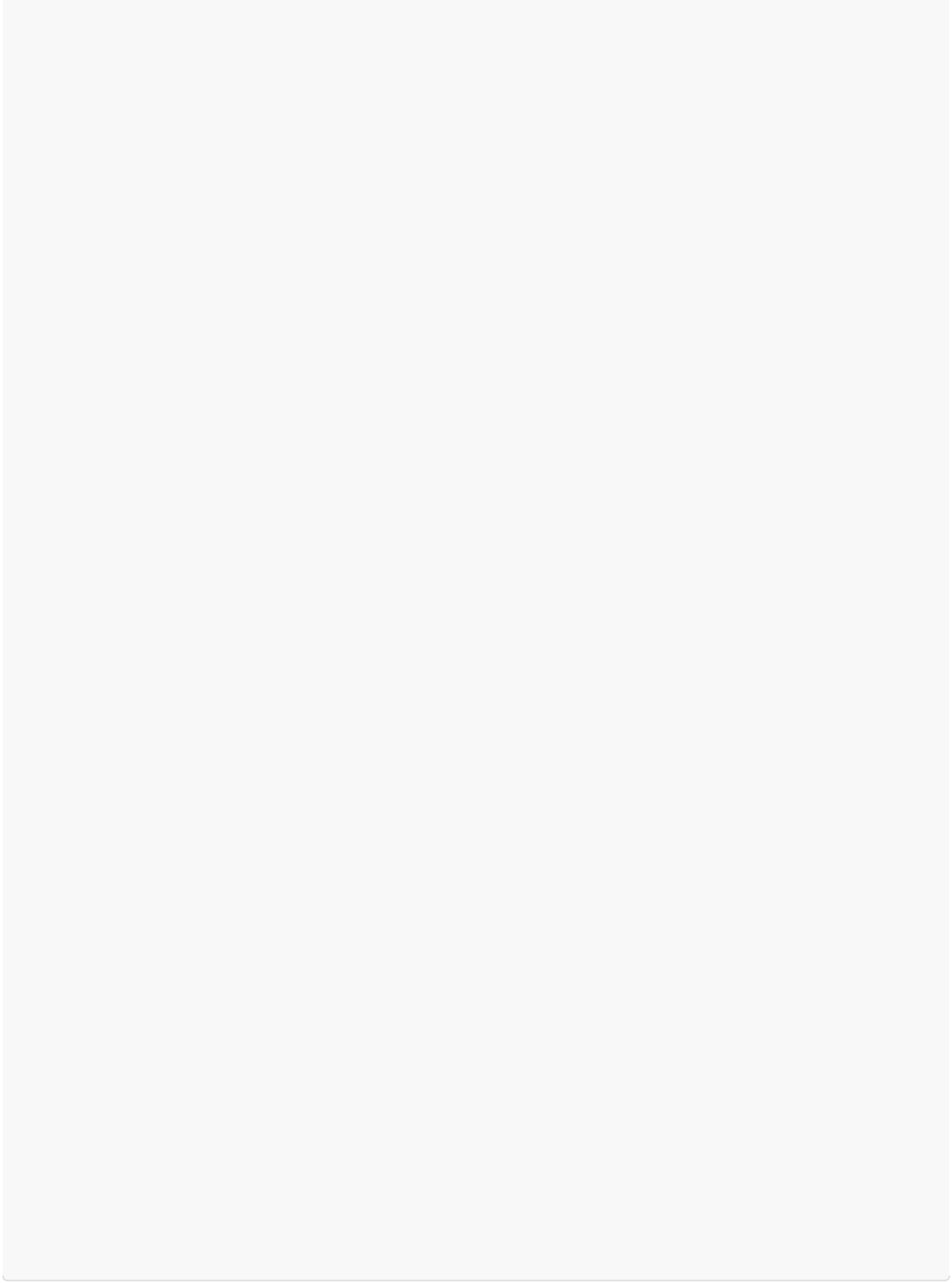
</Menu.Item>

<Menu.Item key="3">

<Icon type="inbox"/>

<span>Option 3</span>

</Menu.Item>



<SubMenu key="sub1" title={<span><Icon

type="mail"/><span>Navigation One</span></span>}>

<Menu.Item key="5">Option 5</Menu.Item> <Menu.Item key="6">Option 6</Menu.Item> <Menu.Item key="7">Option 7</Menu.Item> <Menu.Item key="8">Option 8</Menu.Item>

</SubMenu>

<SubMenu key="sub2" title={<span><Icon

type="appstore"/><span>Navigation Two</span></span>}>

<Menu.Item key="9">Option 9</Menu.Item> <Menu.Item key="10">Option 10</Menu.Item> <SubMenu key="sub3" title="Submenu">

<Menu.Item key="11">Option 11</Menu.Item> <Menu.Item key="12">Option 12</Menu.Item>

</SubMenu>

</SubMenu>

</Menu>

</Sider>

<Layout>

<Header style={{background: '#fff', textAlign:

'center', padding: 0}}>Header</Header>

<Content style={{margin: '24px 16px 0'}}>

<div style={{padding: 24, background: '#fff',

minHeight: 360}}>

{this.props.children}

</div>

</Content>

<Footer style={{textAlign: 'center'}}>后台系统 ©2018 Created by 黑马程序员</Footer>

</Layout>

</Layout>

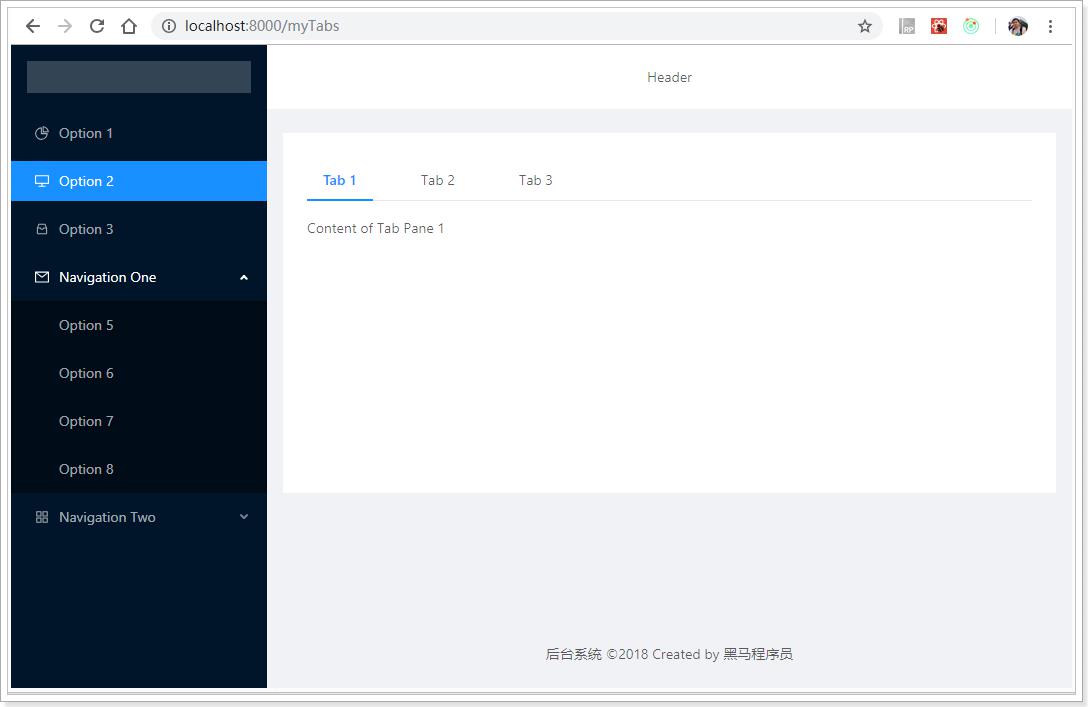
);

}

}

export default BasicLayout;

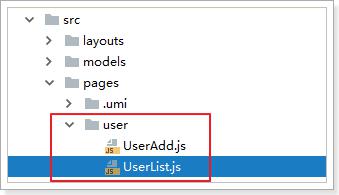
效果：



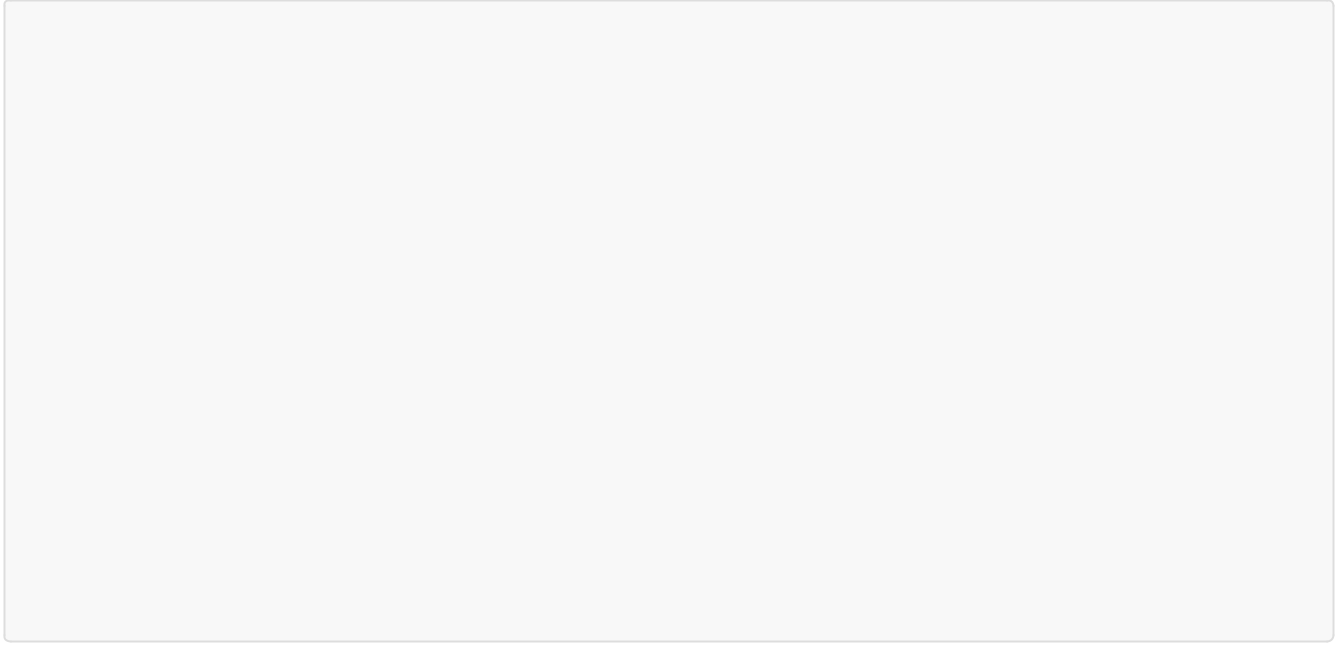
2.3.6、为导航添加链接

下面，我们对左侧的导航条添加链接，点击相应的链接在右侧进行相应页面的显示。

在src目录下创建user目录，并且在user目录下创建UserAdd.js和UserList.js文件，用于模拟实现新增用户和查询用户列表功能。



UserAdd.js：

import React from 'react'

class UserAdd extends React.Component{

render(){

return (

<div>新增用户</div>

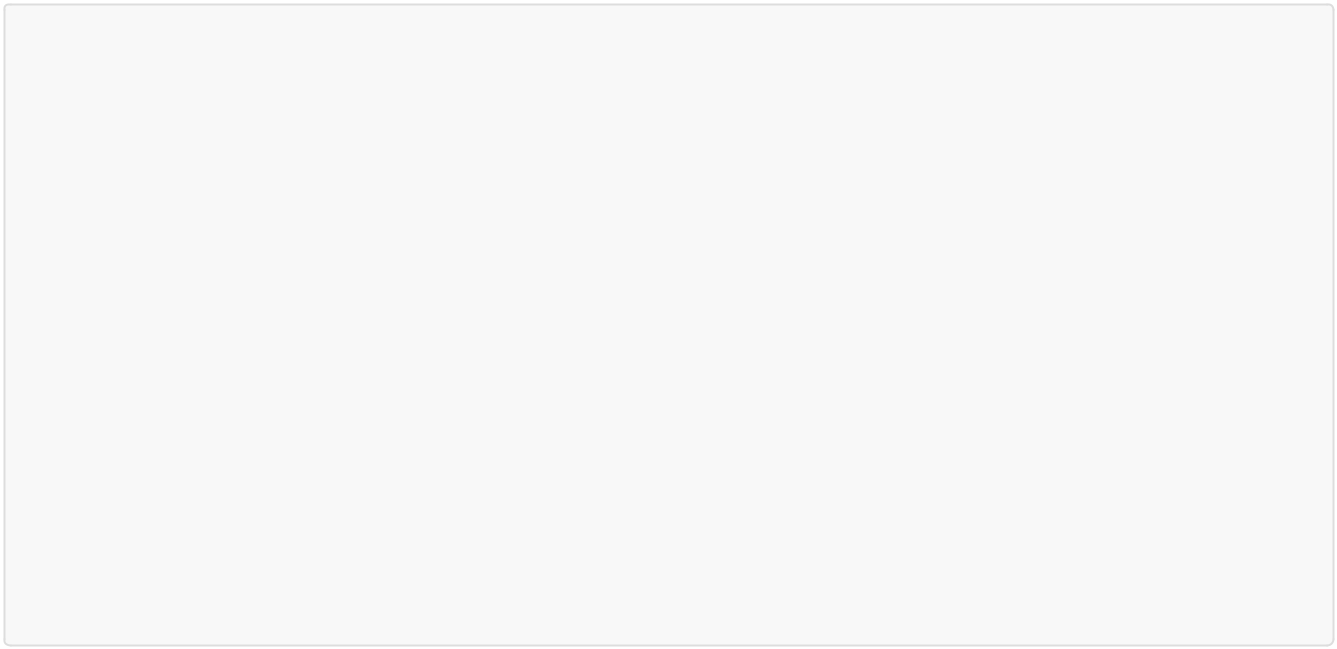
);

}

}

export default UserAdd;

UserList.js：



import React from 'react'

class UserList extends React.Component{

render(){

return (

<div>用户列表</div>

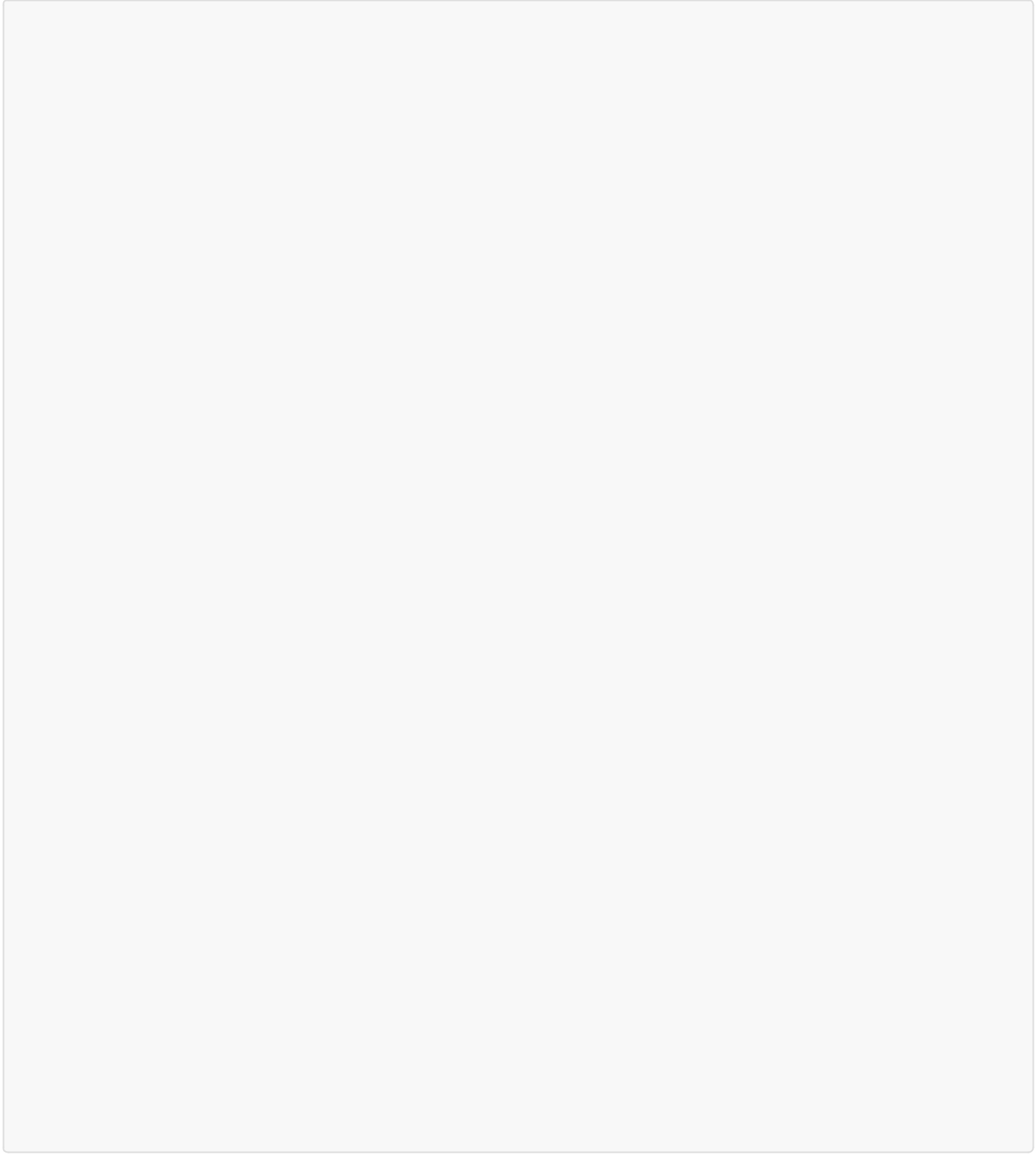
);

}

}

export default UserList;

接下来，配置路由：

export default {

plugins: [

['umi‐plugin‐react', {

dva: true, // 开启dva功能

antd: true // 开启Ant Design功能

}]

],

routes: [{

path: '/',

component: '../layouts', //配置布局路由

routes: [

{

path: '/myTabs',

component: './myTabs'

},

{

path: '/user',

routes: [

{

path: '/user/list',

component: './user/UserList'

},

{

path: '/user/add',

component: './user/UserAdd'

}

]

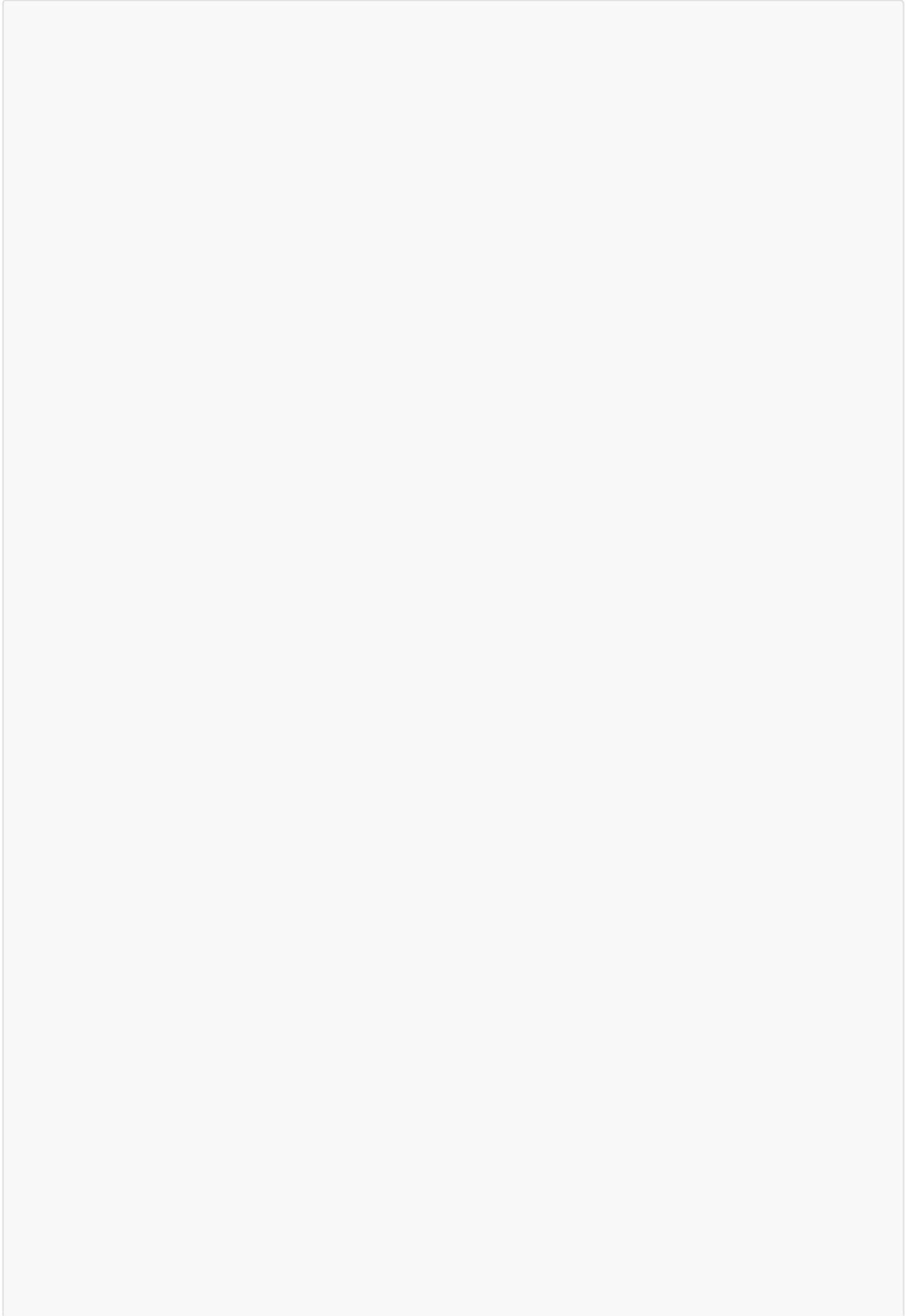
}

]

}]

};

为菜单添加链接：

import React from 'react'

import {Layout, Menu, Icon} from 'antd'; import Link from 'umi/link';

const {Header, Footer, Sider, Content} = Layout; const SubMenu = Menu.SubMenu;

class BasicLayout extends React.Component {

constructor(props){

super(props);

this.state = {

collapsed: false,

}

}

render() {

return (

<Layout>

<Sider width={256} style={{minHeight: '100vh', color:

'white'}}>

<div style={{ height: '32px', background:

'rgba(255,255,255,.2)', margin: '16px'}}/> <Menu

defaultSelectedKeys={['1']}

defaultOpenKeys={['sub1']}

mode="inline"

theme="dark"

inlineCollapsed={this.state.collapsed}

>

<SubMenu key="sub1" title={<span><Icon

type="user"/><span>用户管理</span></span>}>

<Menu.Item key="1">

<Link to="/user/list">用户列表</Link> </Menu.Item>

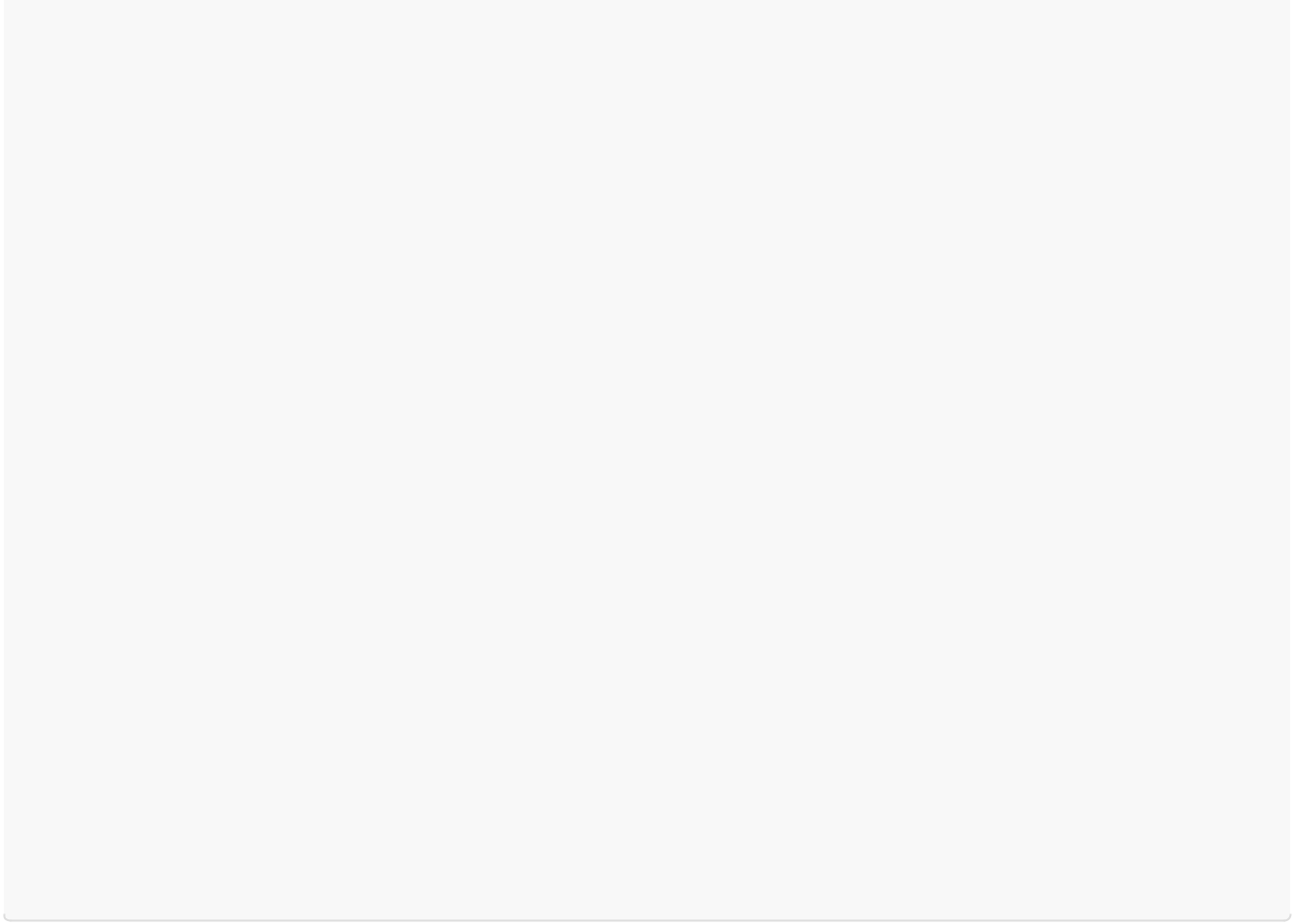
<Menu.Item key="2">

<Link to="/user/add">新增用户</Link> </Menu.Item>

</SubMenu>

</Menu>

</Sider>



<Layout>

<Header style={{background: '#fff', textAlign:

'center', padding: 0}}>Header</Header>

<Content style={{margin: '24px 16px 0'}}>

<div style={{padding: 24, background: '#fff',

minHeight: 360}}>

{this.props.children}

</div>

</Content>

<Footer style={{textAlign: 'center'}}>后台系统 ©2018 Created by 黑马程序员</Footer>

</Layout>

</Layout>

);

}

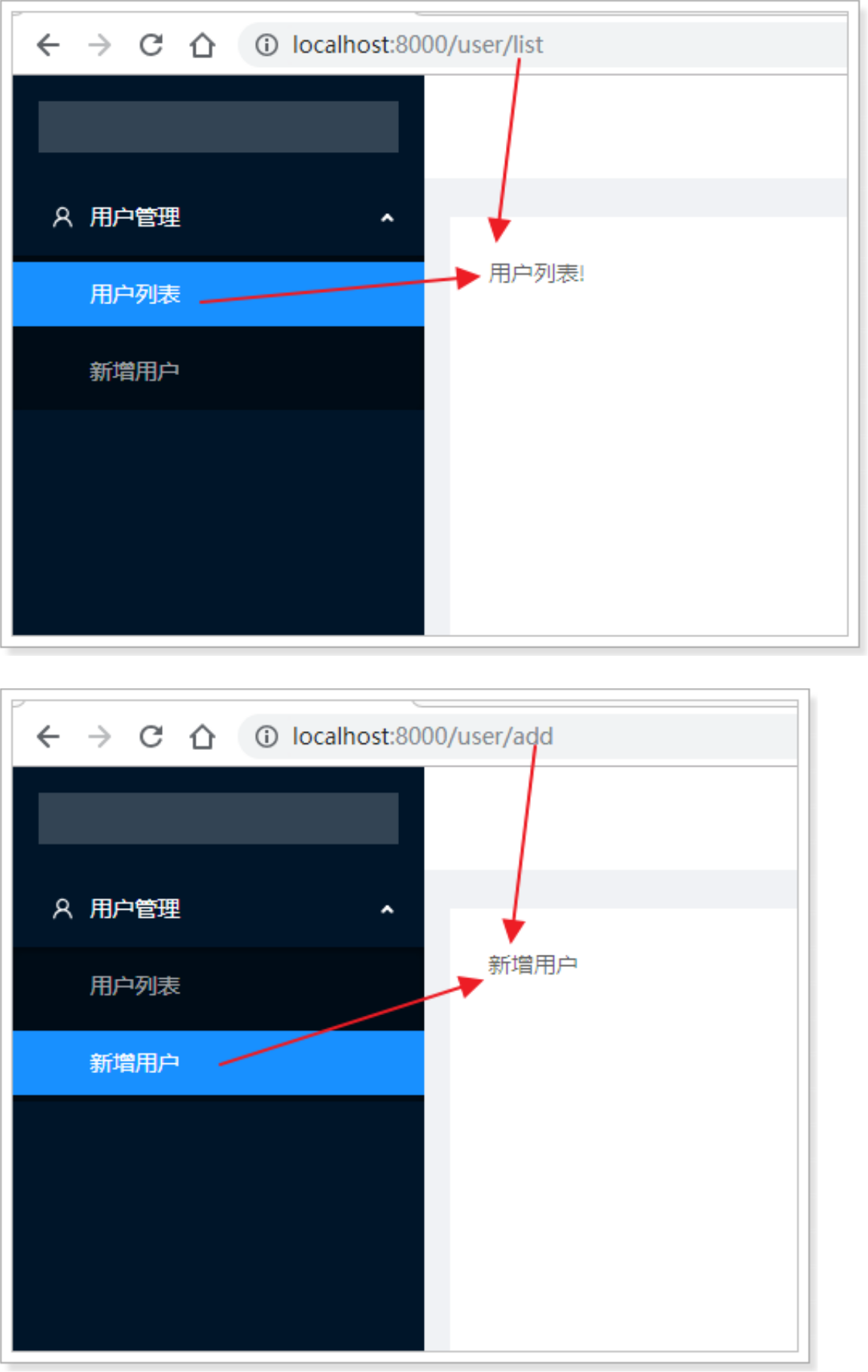
}

export default BasicLayout;



注意：这里使用了umi的link标签，目的是出现记录点击的菜单。

测试：

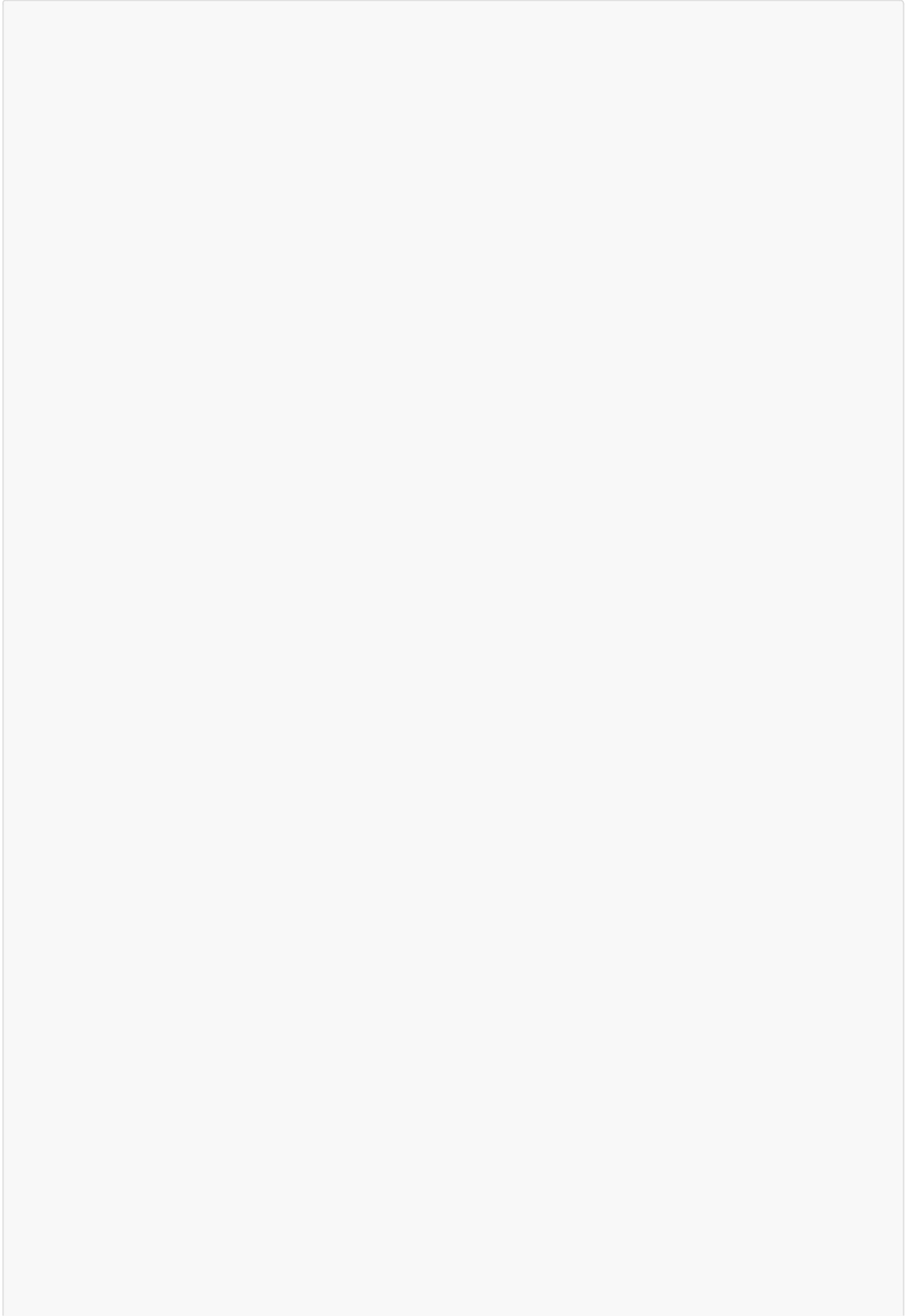


2.4、表格

2.4.1、基本用法

参考官方文档进行实现：<https://ant.design/components/table-cn/>

改造UserList.js页面：

import React from 'react'

import {Table, Divider, Tag, Pagination } from 'antd';

const {Column} = Table;

const data = [{

key: '1',

name: '张三',

age: 32,

address: '上海市',

tags: ['程序员', '帅气'],

}, {

key: '2',

name: '李四',

age: 42,

address: '北京市',

tags: ['屌丝'],

}, {

key: '3',

name: '王五',

age: 32,

address: '杭州市',

tags: ['高富帅', '富二代'],

}];

class UserList extends React.Component {

render() {

return (

<div>

<Table dataSource={data} pagination=

{{position:"bottom",total:500,pageSize:10, defaultCurrent:3}}>

<Column

title="姓名"

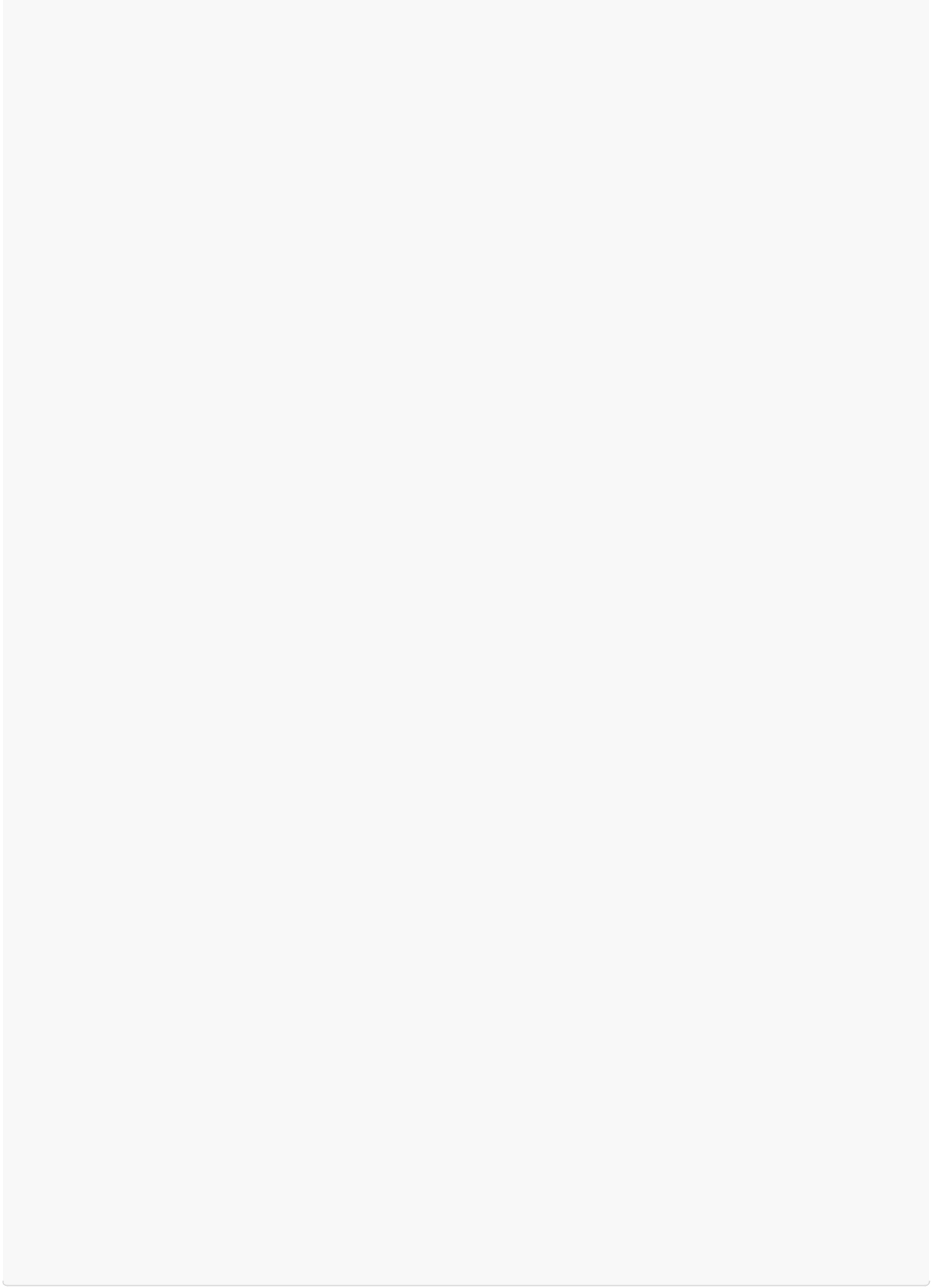
dataIndex="name"

key="name"

/>

<Column

title="年龄"



dataIndex="age"

key="age"

/>

<Column

title="地址"

dataIndex="address"

key="address"

/>

<Column

title="标签"

dataIndex="tags"

key="tags"

render={tags => (

<span>

{tags.map(tag => <Tag color="blue" key=

{tag}>{tag}</Tag>)}

</span>

)}

/>

<Column

title="操作"

key="action"

render={(text, record) => (

<span>

<a href="javascript:;">编辑</a>

<Divider type="vertical"/>

<a href="javascript:;">删除</a>

</span>

)}

/>

</Table>

</div>

);

}

}

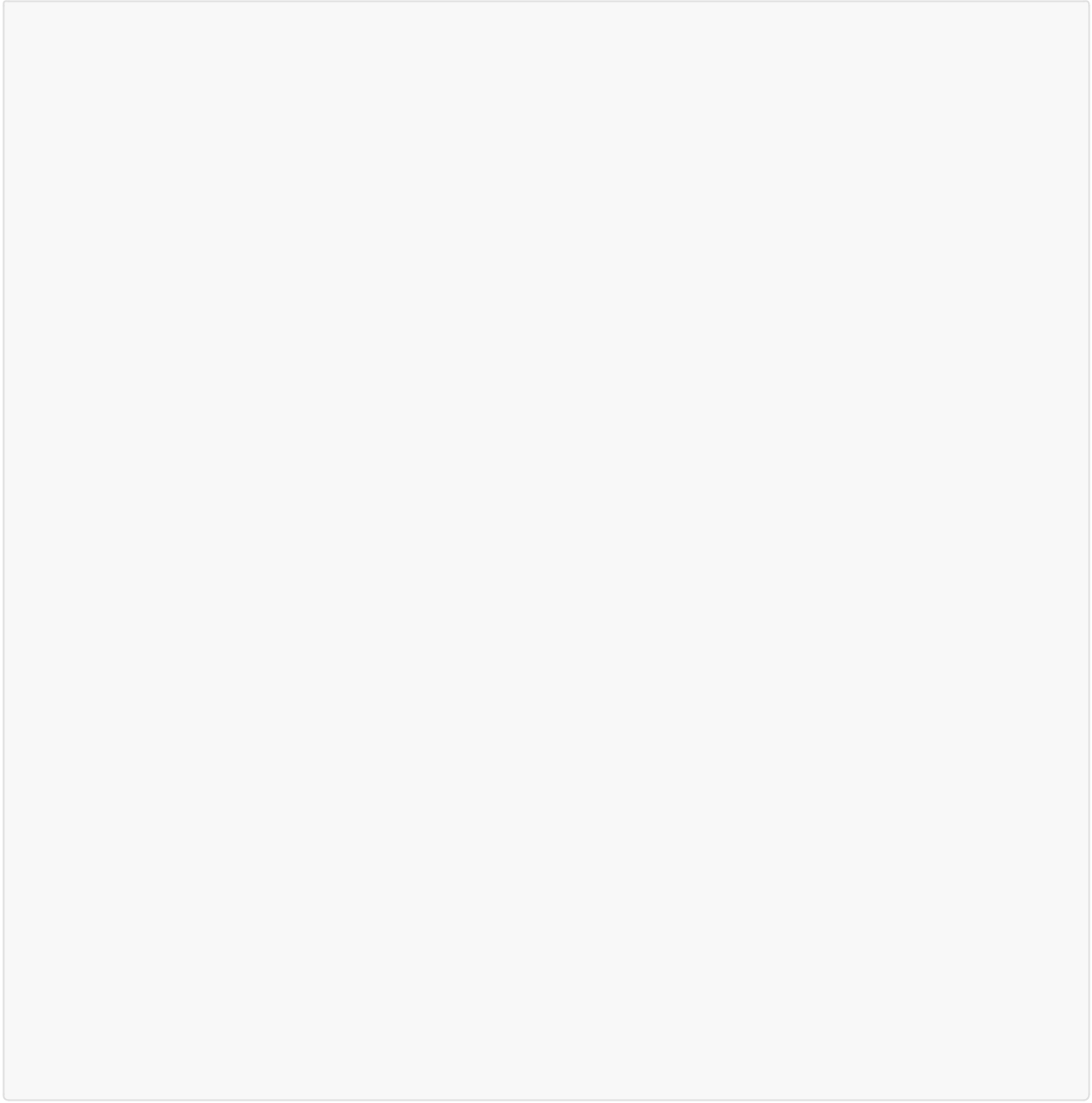
export default UserList;

实现效果：



2.4.2、将数据分离到model中

model的实现：UserListData.js

import request from "../util/request";

export default {

namespace: 'userList',

state: {

list: []

},

effects: {

\*initData(params, sagaEffects) {

const {call, put} = sagaEffects;

const url = "/ds/user/list";

let data = yield call(request, url);

yield put({

type : "queryList",

data : data

});

}

},

reducers: {

queryList(state, result) {

let data = [...result.data];

return { //更新状态值

list: data

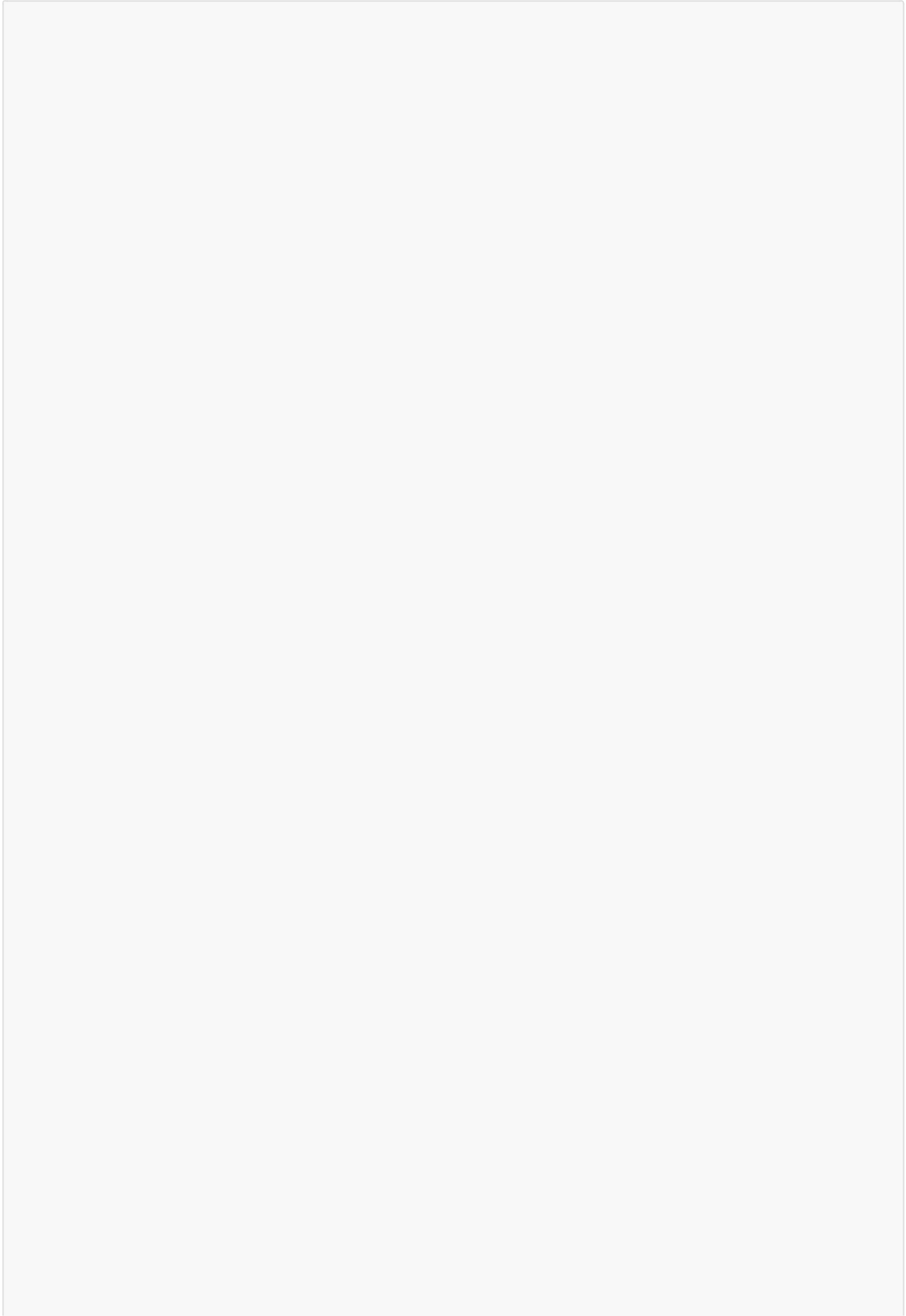
}

}

}

}

修改UserList.js中的逻辑：

import React from 'react';

import { connect } from 'dva';

import {Table, Divider, Tag, Pagination } from 'antd';

const {Column} = Table;

const namespace = 'userList';

@connect((state)=>{

return {

data : state[namespace].list

}

}, (dispatch) => {

return {

initData : () => {

dispatch({

type: namespace + "/initData"

});

}

}

})

class UserList extends React.Component {

componentDidMount(){

this.props.initData();

}

render() {

return (

<div>

<Table dataSource={this.props.data} pagination=

{{position:"bottom",total:500,pageSize:10, defaultCurrent:3}}> <Column

title="姓名"

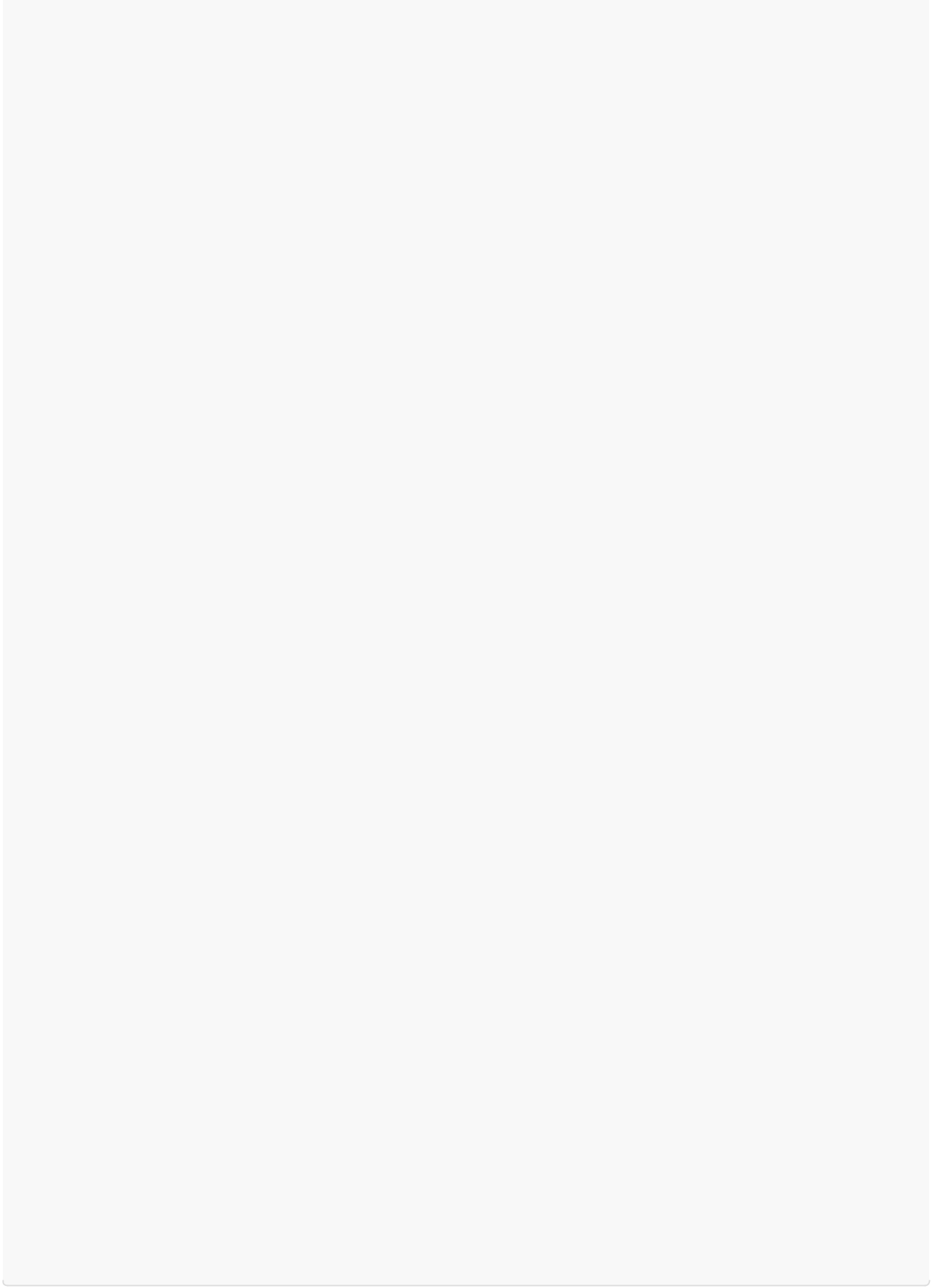
dataIndex="name"

key="name"

/>

<Column

title="年龄"



dataIndex="age"

key="age"

/>

<Column

title="地址"

dataIndex="address"

key="address"

/>

<Column

title="标签"

dataIndex="tags"

key="tags"

render={tags => (

<span>

{tags.map(tag => <Tag color="blue" key=

{tag}>{tag}</Tag>)}

</span>

)}

/>

<Column

title="操作"

key="action"

render={(text, record) => (

<span>

<a href="javascript:;">编辑</a>

<Divider type="vertical"/>

<a href="javascript:;">删除</a>

</span>

)}

/>

</Table>

</div>

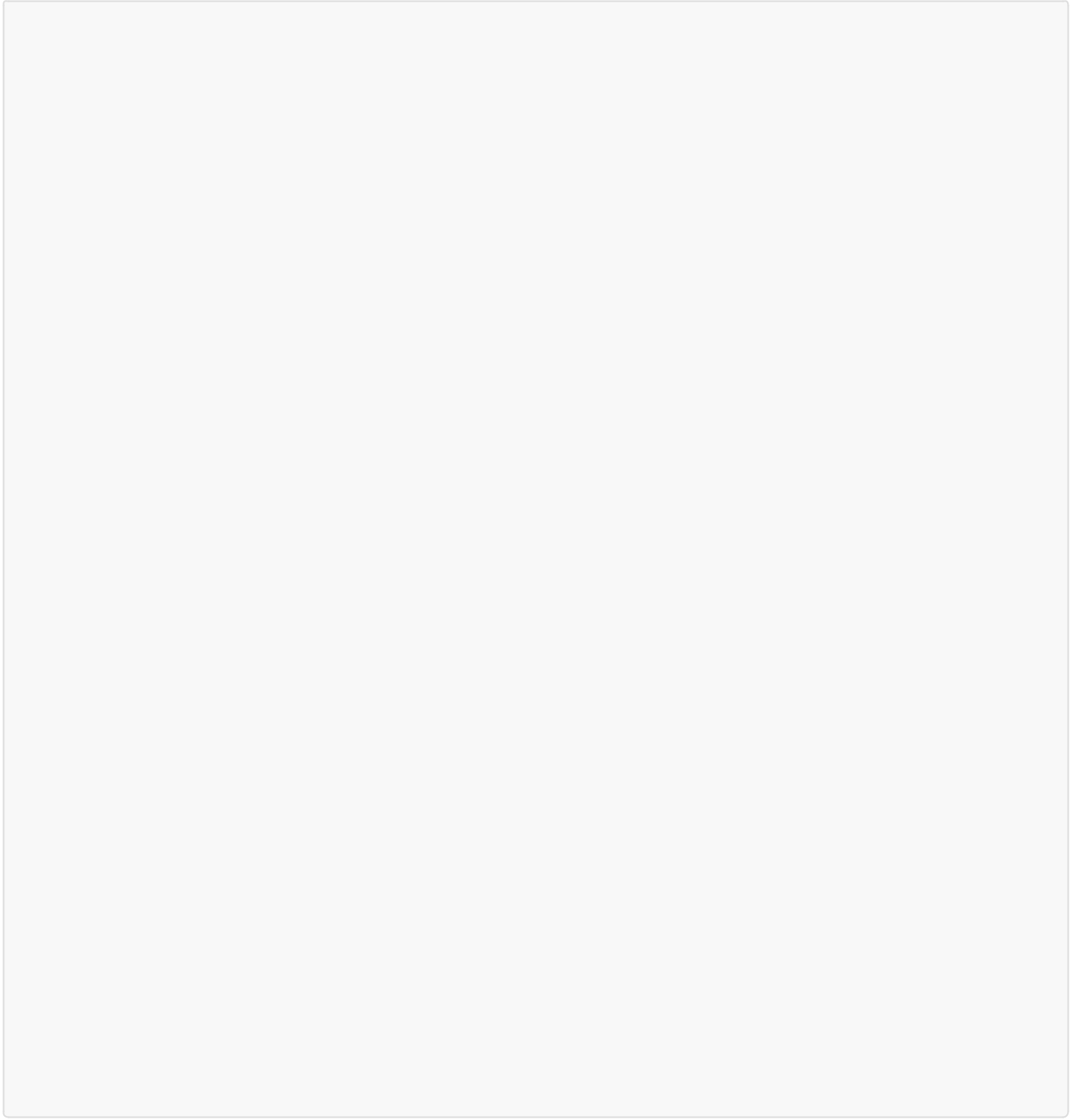
);

}

}

export default UserList;

mock数据：MockListData.js

export default {

'get /ds/list': function (req, res) {

res.json({

data: [1, 2, 3, 4],

maxNum: 4

});

},

'get /ds/user/list': function (req, res) { res.json([{

key: '1',

name: '张三1',

age: 32,

address: '上海市',

tags: ['程序员', '帅气'],

}, {

key: '2',

name: '李四',

age: 42,

address: '北京市',

tags: ['屌丝'],

}, {

key: '3',

name: '王五',

age: 32,

address: '杭州市',

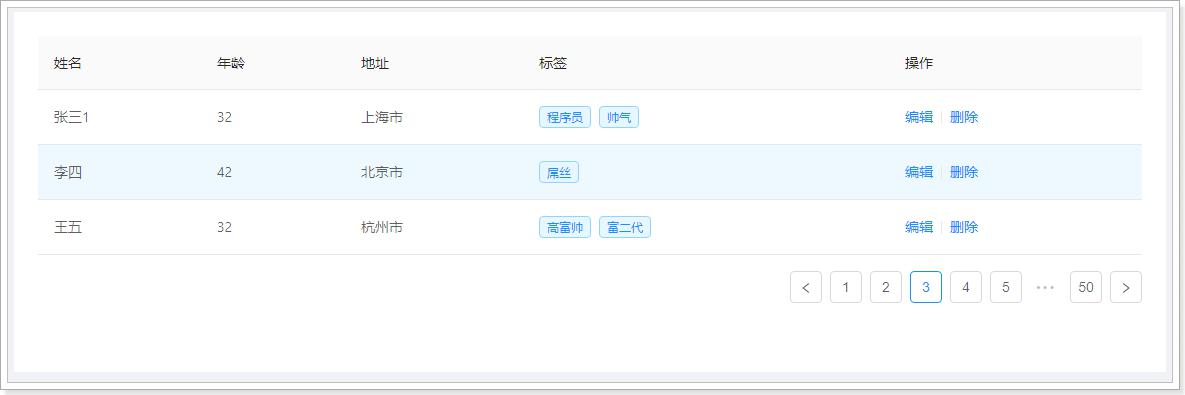
tags: ['高富帅', '富二代'],

}]);

}

}

测试结果：



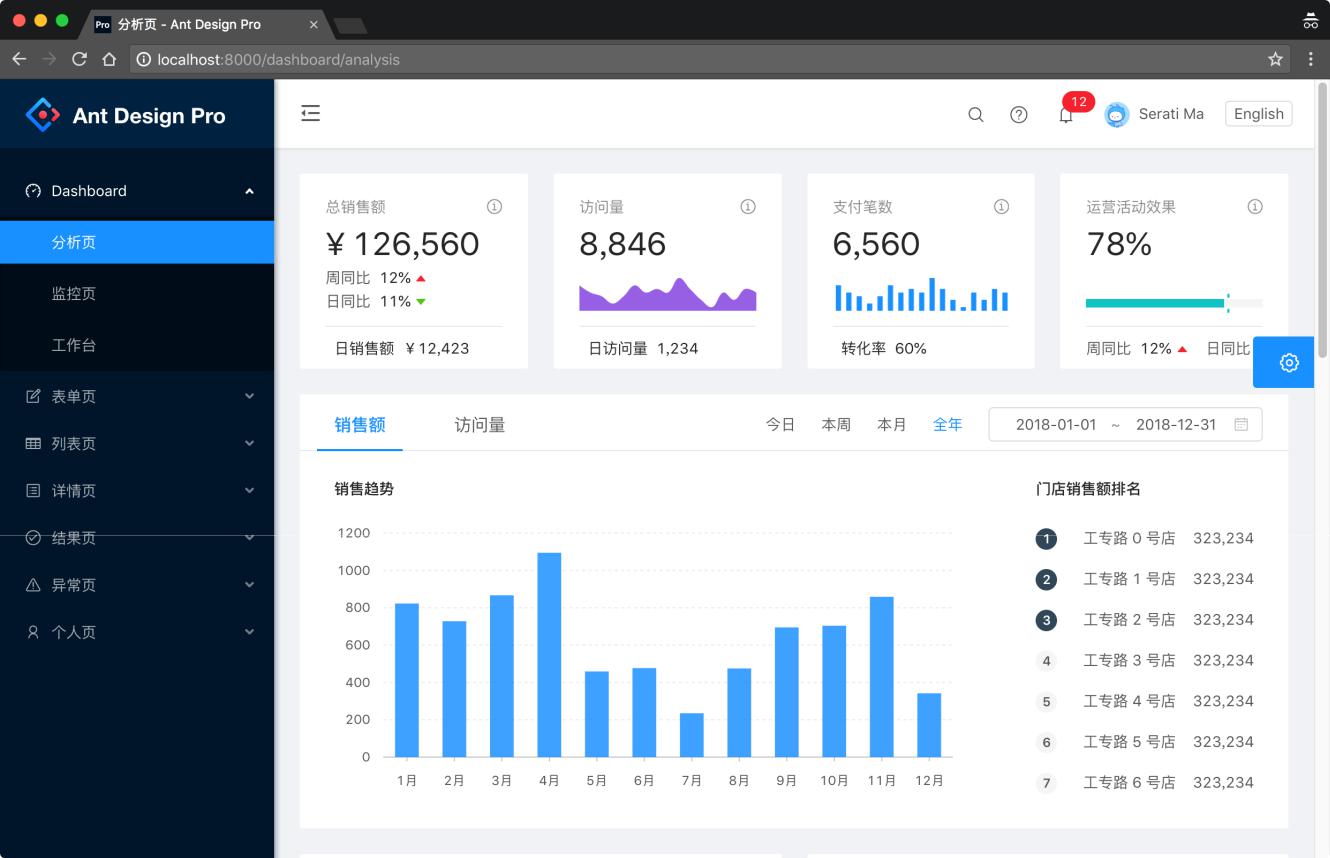
3、Ant Design Pro入门



3.1、了解Ant Design Pro

Ant Design Pro 是基于Ant Design的一个开箱即用的，企业级中后台前端/设计解决方案。

效果：



源码地址：<https://github.com/ant-design/ant-design-pro>

特性：



3.2、快速入门

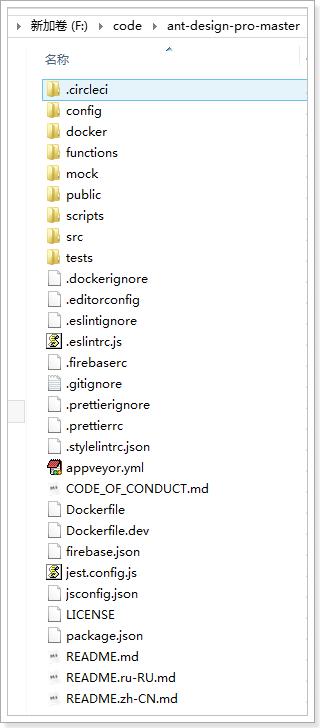
3.2.1、部署安装

下载地址：<https://github.com/ant-design/ant-design-pro>

我们使用资料中提供的，已经下载好的文件：ant-design-pro-master.zip

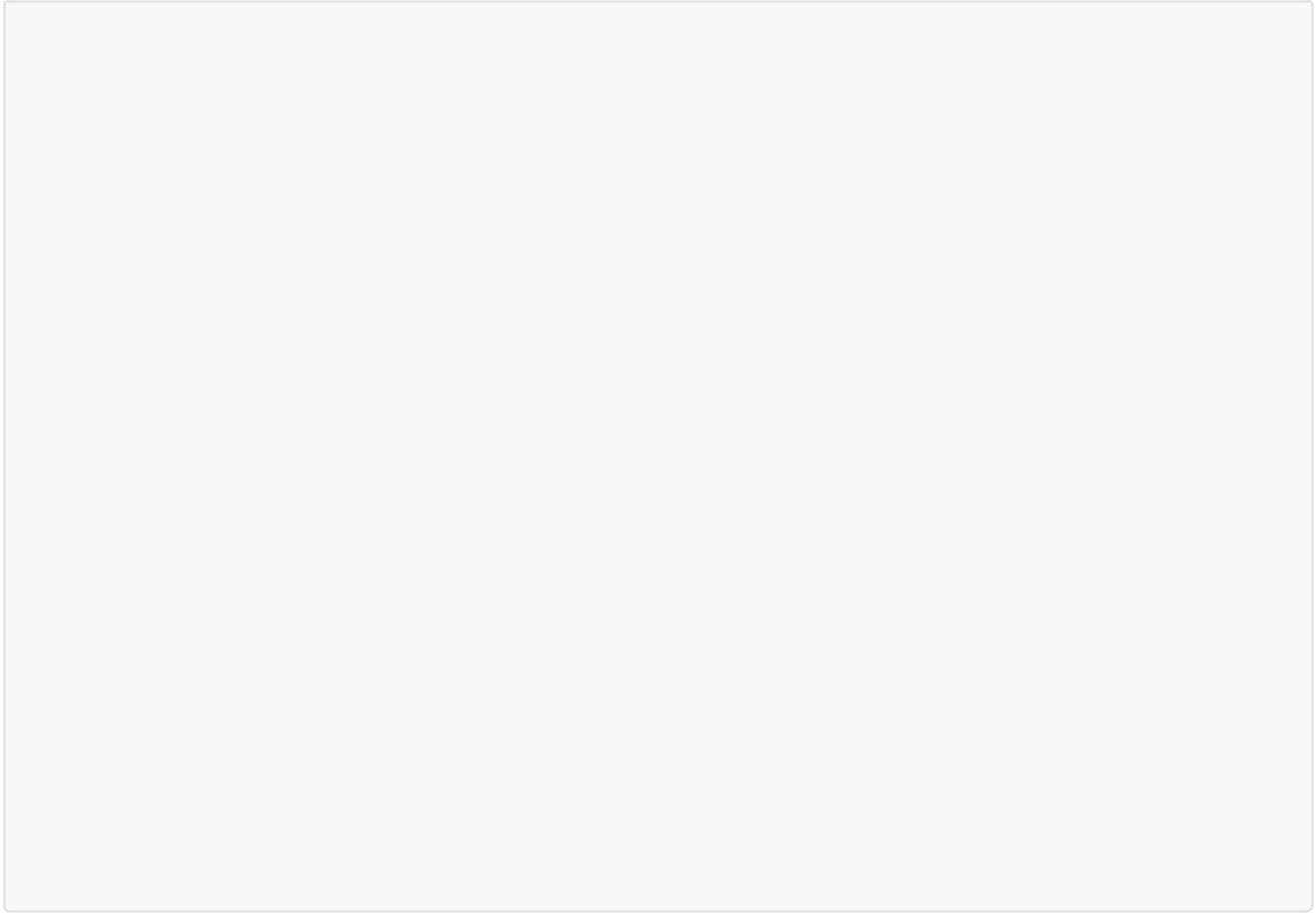


第一步：将ant-design-pro-master.zip解压到任意目录，我的目录是F:\code

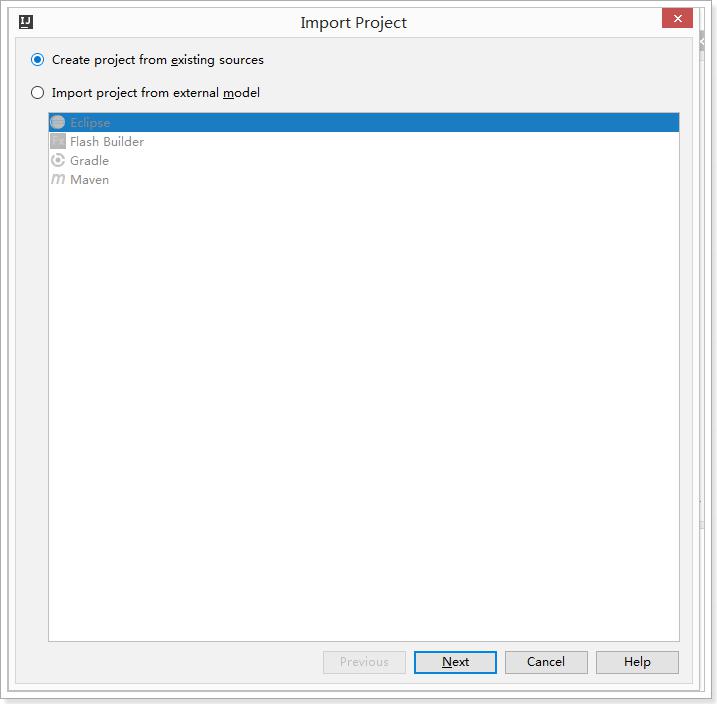


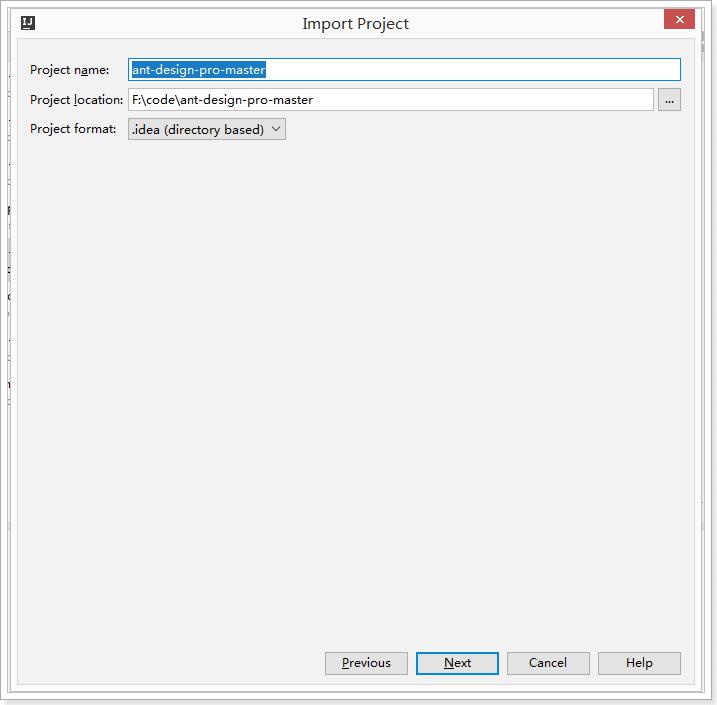
Ant Design Pro提供的目录如下：

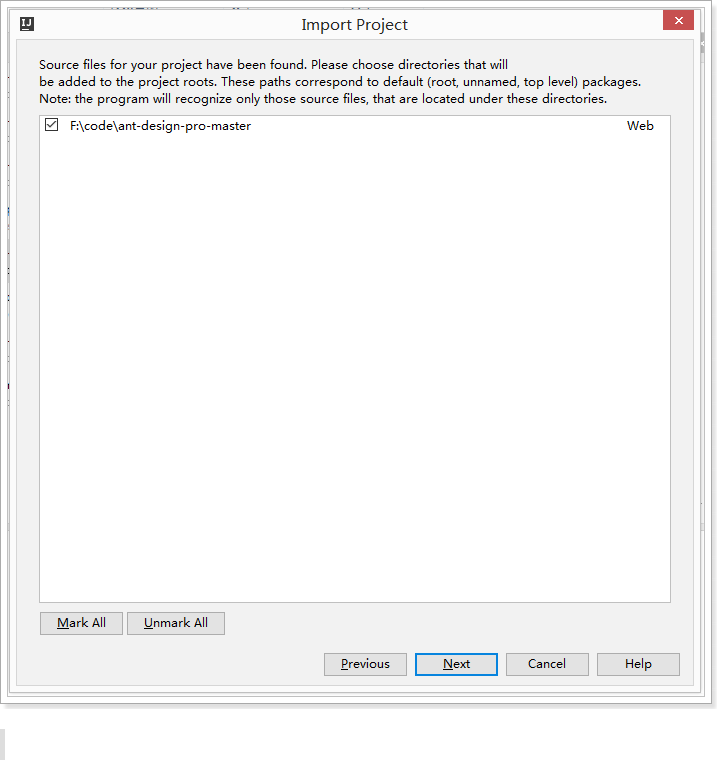
|  |  |  |
| --- | --- | --- |
| ├── config | | # umi 配置，包含路由，构建等配置 |
| ├── mock | | # 本地模拟数据 |
| ├── public | |  |
| │ | └── favicon.png | # Favicon |
| ├── src | |  |
| │ | ├── assets | # 本地静态资源 |
| │ | ├── components | # 业务通用组件 |
| │ | ├── e2e | # 集成测试用例 |
| │ | ├── layouts | # 通用布局 |
| │ | ├── models | # 全局 dva model |
| │ | ├── pages | # 业务页面入口和常用模板 |
| │ | ├── services | # 后台接口服务 |
| │ | ├── utils | # 工具库 |
| │ | ├── locales | # 国际化资源 |
| │ | ├── global.less | # 全局样式 |
| │ | └── global.js | #全局JS |
| ├── tests | | # 测试工具 |
| ├── README.md | |  |
| └── package.json | |  |



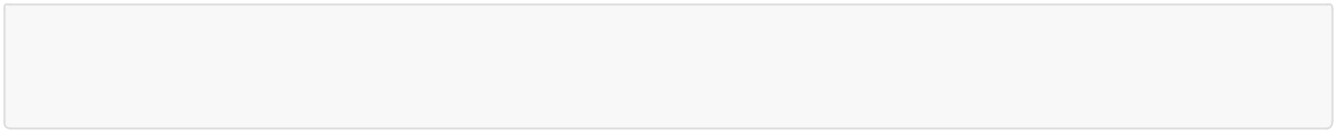
第二步，导入项目到Idea中







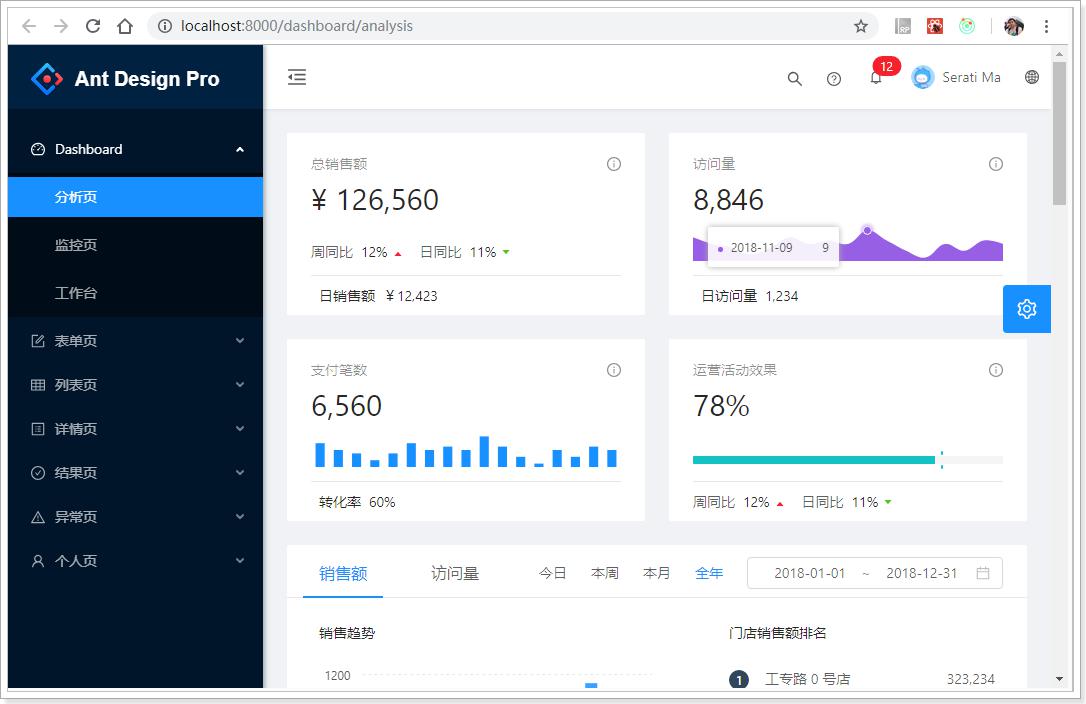
第三步：进行初始化以及启动



tyarn install #安装相关依赖

tyarn start #启动服务

测试：

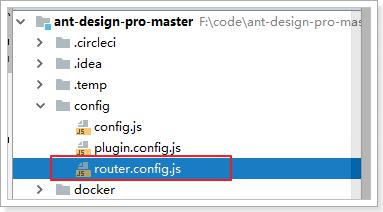


可以看到，系统已经启动完成。

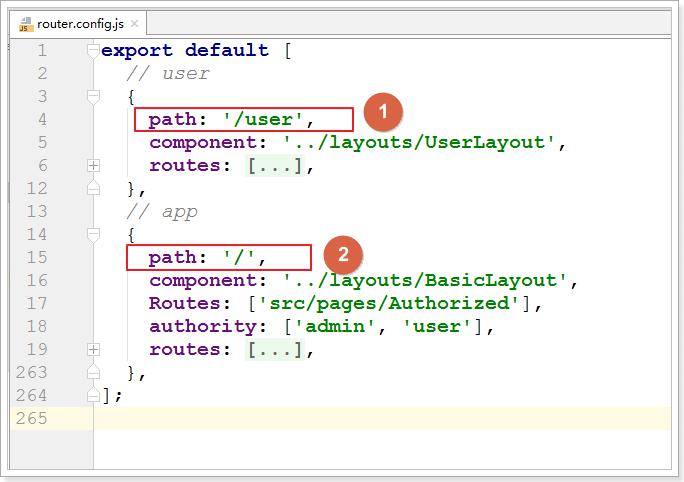
3.2.2、菜单和路由

默认的菜单是不能直接投入到项目开发的，所以，我们需要搞清楚如何自定义菜单和路由。

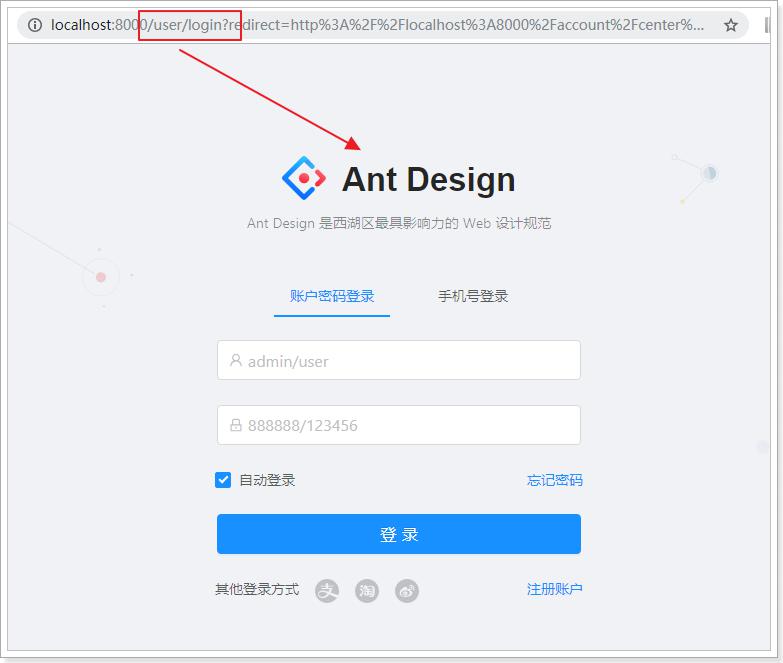
在pro中，菜单和路由，在router.config.js配置文件中进行管理：



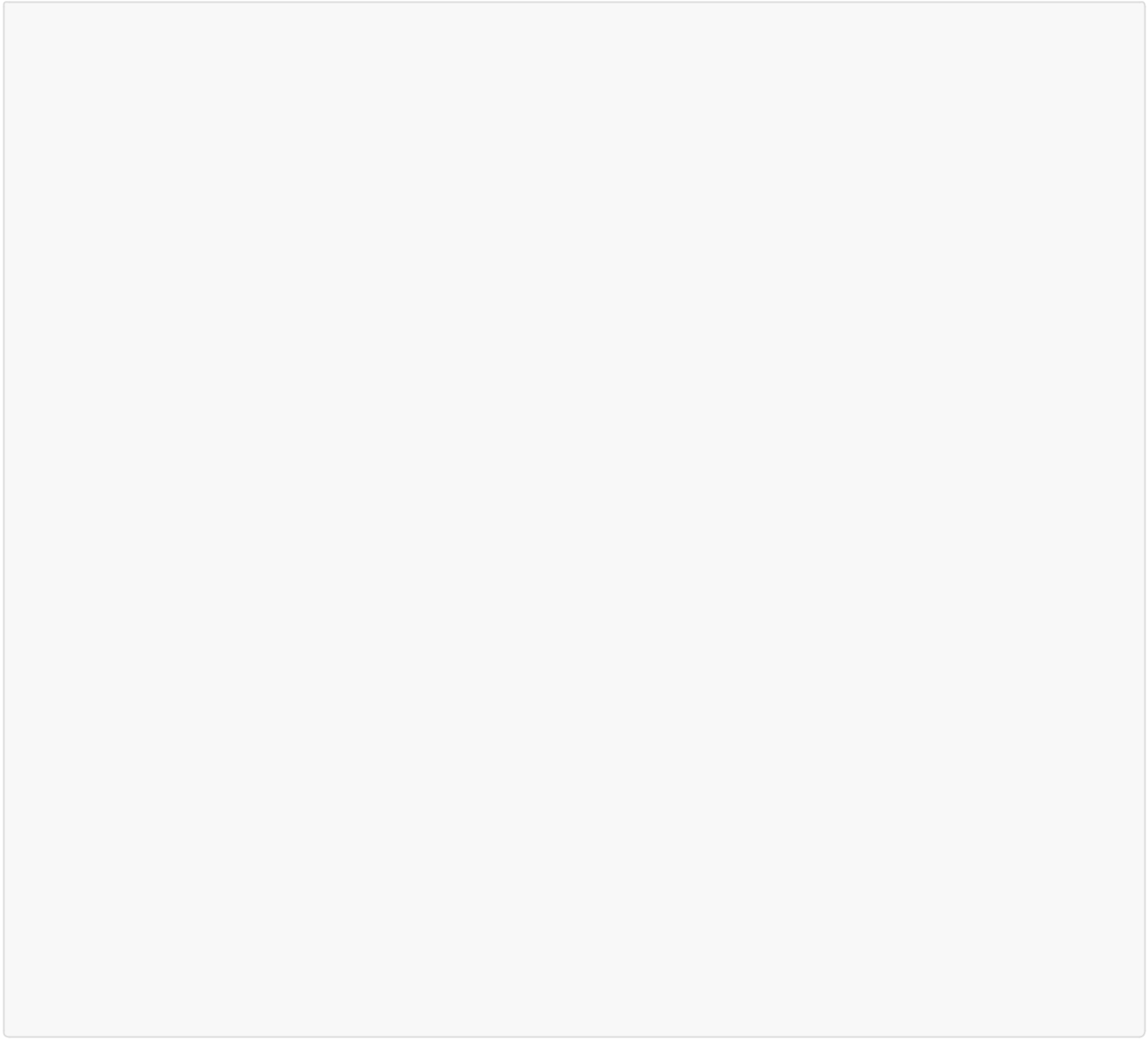
打开router.config.js后，可以看出，pro提供了有2套路由（布局），分别是/user和/



/user的布局：



接下来，我们先重点关注，/路由：

routes: [

// dashboard

{ path: '/', redirect: '/dashboard/analysis' },

{

path: '/dashboard',

name: 'dashboard',

icon: 'dashboard',

routes: [

{

path: '/dashboard/analysis', //请求路径

name: 'analysis',

component: './Dashboard/Analysis', //组件位置 },

{

path: '/dashboard/monitor',

name: 'monitor',

component: './Dashboard/Monitor', },

{

path: '/dashboard/workplace',

name: 'workplace',

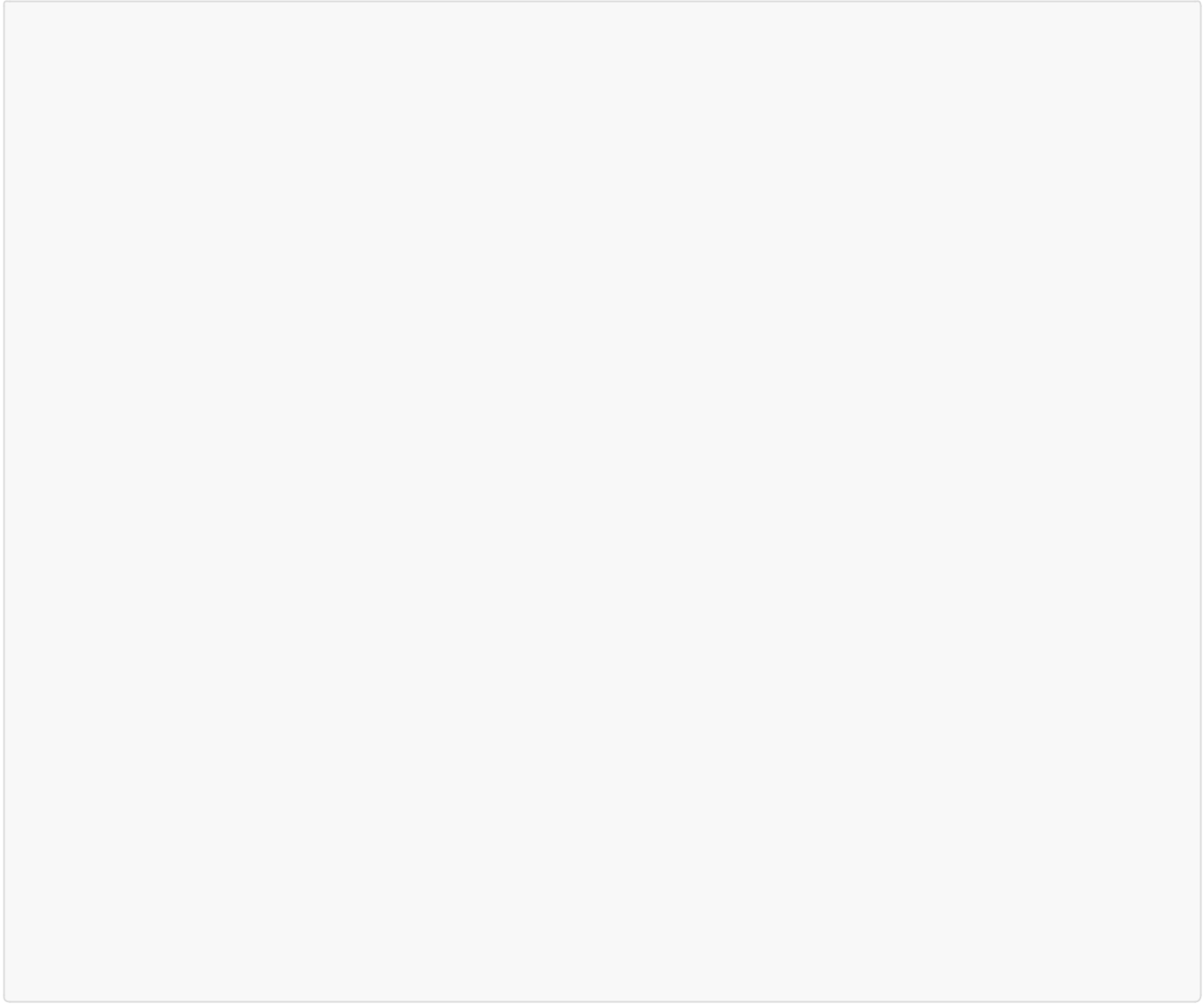
component: './Dashboard/Workplace', },

],

},

所以，可以得出结论，菜单是有路由的配置生成的。

接下来进行实验，新增一个路由：

// new

{

path: '/new',

name: 'new',

icon: 'user',

routes: [

{

path: '/new/analysis',

name: 'analysis',

component: './Dashboard/Analysis',

},

{

path: '/new/monitor',

name: 'monitor',

component: './Dashboard/Monitor',

},

{

path: '/new/workplace',

name: 'workplace',

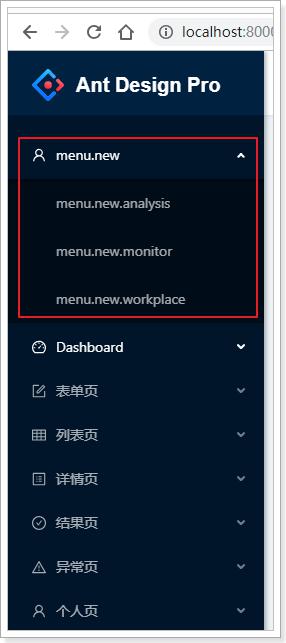
component: './Dashboard/Workplace',

},

],

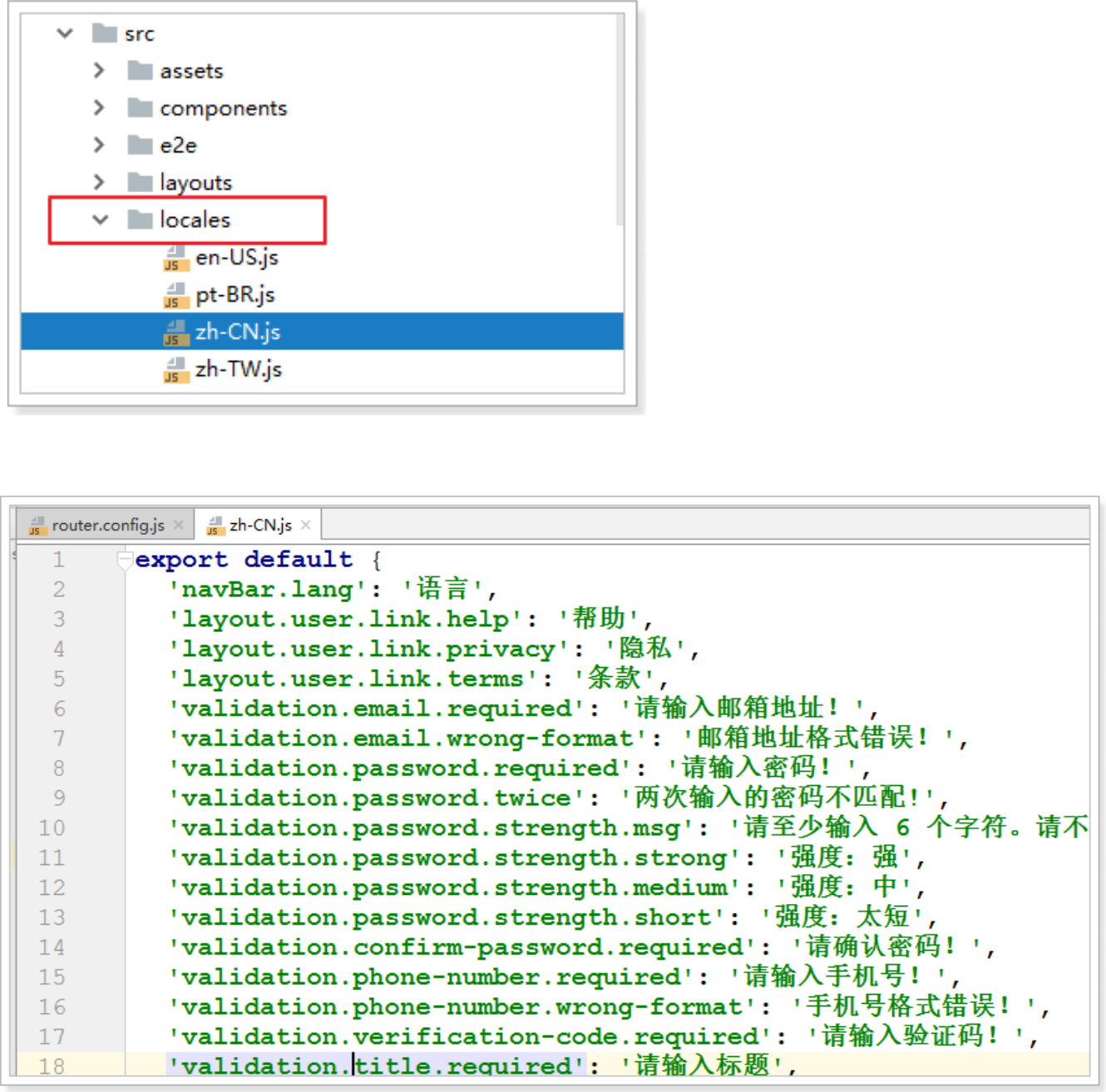
},

测试：

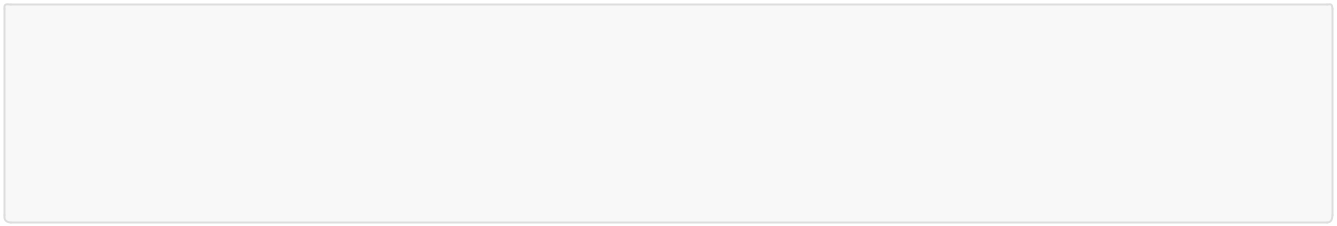


可以看出，新的菜单以及添加到页面中，只是显示的文字不对。那么文字在哪里配置呢？

其实，文字是在国际化文件中进行配置的：



新增配置如下：



'menu.new': 'New Dashboard',

'menu.new.analysis': 'New 分析页',

'menu.new.monitor': 'New 监控页',

'menu.new.workplace': 'New 工作台',

进行测试：

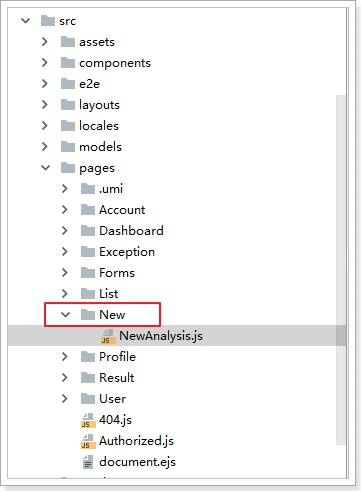


发现，已经正常显示了。

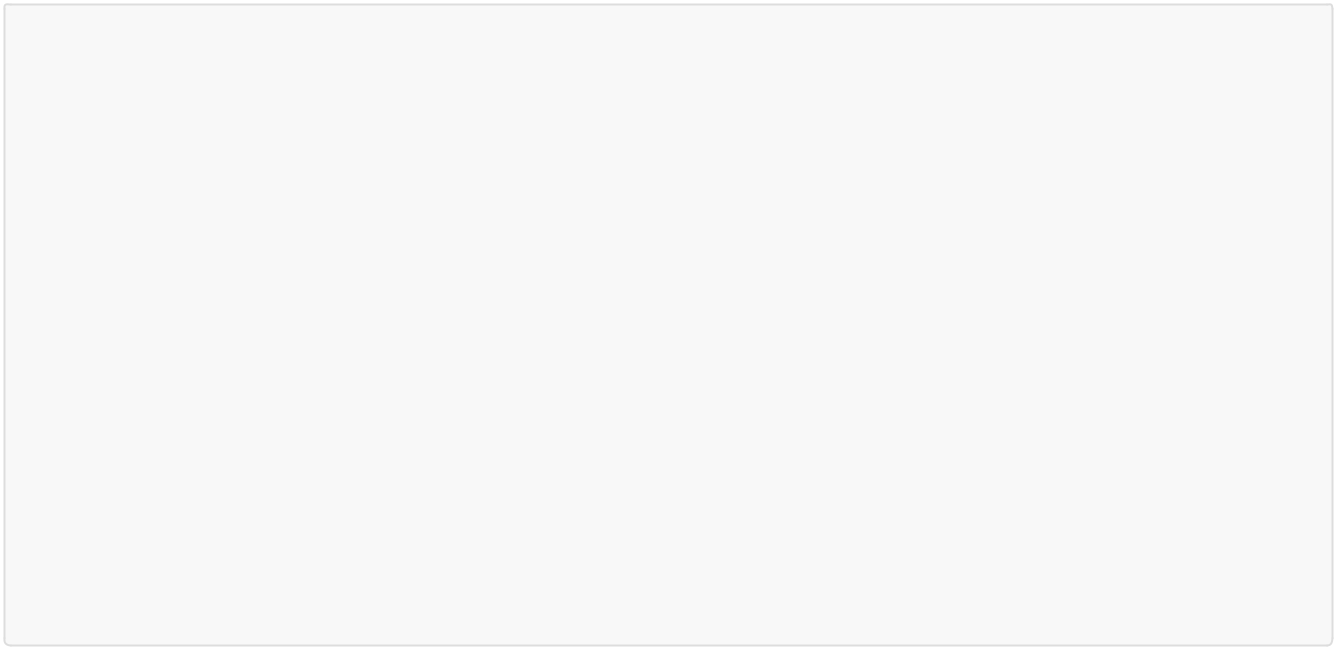
3.2.3、新增页面

上面我们添加了新的菜单，但是页面依然使用的是模板中的页面，那么如何新增页面呢？

所有的页面依然是保存的src/pages中，在pages目录下，以功能为单元创建目录，如：



创建文件 NewAnalysis.js：



import React from 'react';

class NewAnalysis extends React.Component {

render() {

return (

<div>NewAnalysis</div>

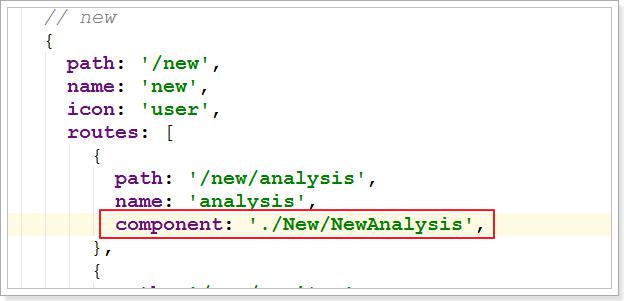
);

}

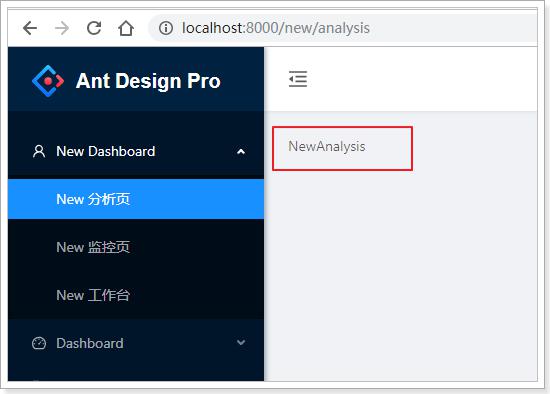
}

export default NewAnalysis;

修改路由中的路径：



测试：



可以看到，一个新的页面就创建好了，并且已经加入到菜单中。

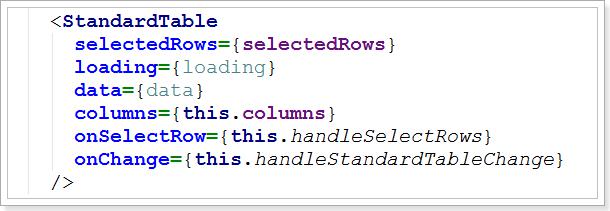
3.2.4、pro中的model执行流程

在pro系统中，model是如何执行的，下面我们以表格为例，探究下在Pro中的执行流程。

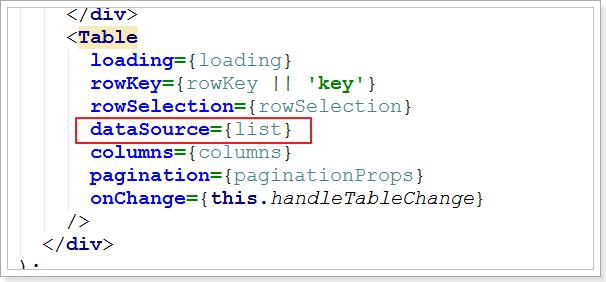


进入TableList.js代码进行查看：

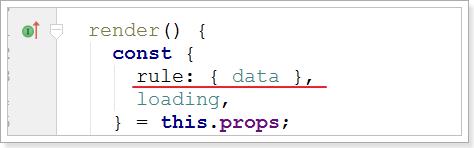
生成表格的主要逻辑在这里：



在StandardTable中，使用Table组件生成表格，其中数据源是data：

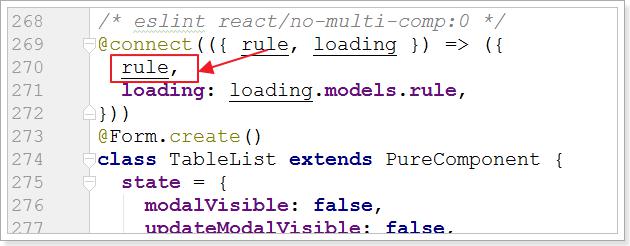


TableList.js中，data数据从构造方法中获取到：

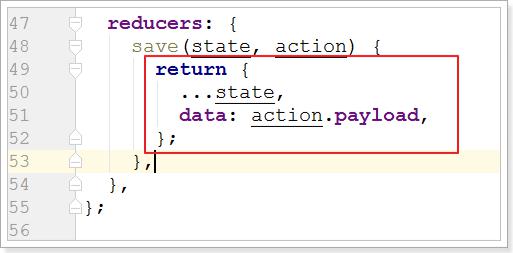


this.props中的rule数据，是通过@connect()修饰器获取：

需要注意的是：{ rule, loading }是解构表达式，从props中获取数据



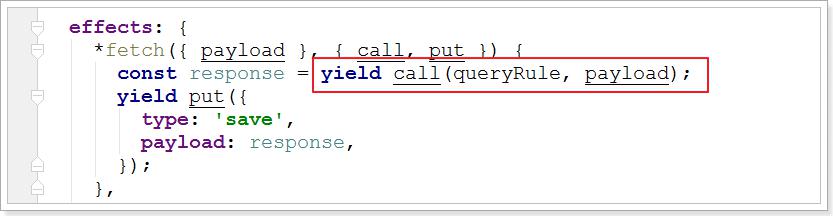
数据从model中获取，在models下的rule.js中：



在TableList.js中，组件加载完成后进行加载数据:



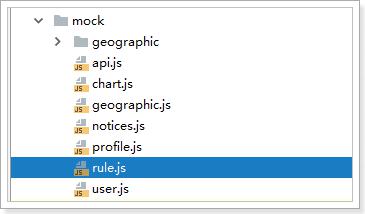
在rule.js中，进行加载数据：



queryRule是在/services/api中进行了定义：



数据的mock是在mock/rule.js中完成。



这就是整个数据的加载、更新流程。