# 之前的笔记链接

react全家桶学习笔记1：[路由基础配置](file:///G:\C以及其他教程\Web开发2023\web开发2023视频教程\凡云1\react零基础学习资料+学习笔记\react全家桶笔记\react全家桶学习笔记1-路由基础配置.docx)

react全家桶学习笔记2：[首页布局和菜单渲染](file:///G:\C以及其他教程\Web开发2023\web开发2023视频教程\凡云1\react零基础学习资料+学习笔记\react全家桶笔记\react全家桶学习笔记2-首页布局和菜单渲染.docx)

react全家桶学习笔记3：[二级路由配置](file:///G:\C以及其他教程\Web开发2023\web开发2023视频教程\凡云1\react零基础学习资料+学习笔记\react全家桶笔记\react全家桶学习笔记3-二级路由配置.docx)

react全家桶学习笔记4： [配置菜单默认项](file:///G:\C以及其他教程\Web开发2023\web开发2023视频教程\凡云1\react零基础学习资料+学习笔记\react全家桶笔记\react全家桶学习笔记4-设置菜单默认项.docx)

react全家桶学习笔记5： [axios配置网络请求](file:///G:\C以及其他教程\Web开发2023\web开发2023视频教程\凡云1\react零基础学习资料+学习笔记\react全家桶笔记\react全家桶学习笔记5-axios设置网络请求.docx)

react全家桶学习笔记6：[用户列表渲染](file:///G:\C以及其他教程\Web开发2023\web开发2023视频教程\凡云1\react零基础学习资料+学习笔记\react全家桶笔记\react全家桶学习笔记6-antd用户列表渲染.docx)

react全家桶学习笔记7：[路由传参](file:///G:\C以及其他教程\Web开发2023\web开发2023视频教程\凡云1\react零基础学习资料+学习笔记\react全家桶笔记\react全家桶学习笔记7-路由传参.docx)

react全家桶学习笔记8: [状态机初始化配置](file:///G:\C以及其他教程\Web开发2023\web开发2023视频教程\凡云1\react零基础学习资料+学习笔记\react全家桶笔记\react全家桶学习笔记8-状态机初始配置.docx)

react全家桶学习笔记9: [redux获取分类数据](file:///G:\C以及其他教程\Web开发2023\web开发2023视频教程\凡云1\react零基础学习资料+学习笔记\react全家桶笔记\react全家桶学习笔记9-redux获取分类数据.docx)

react全家桶学习笔记10：[渲染一二级分类数据以及优化](file:///G:\C以及其他教程\Web开发2023\web开发2023视频教程\凡云1\react零基础学习资料+学习笔记\react全家桶笔记\react全家桶学习笔记10-redux渲染一二级分类以及优化处理.docx)，

react全家桶学习笔记11：[redux获取用户数据](react全家桶学习笔记11-获取用户数据.docx)

# react全家桶学习笔记12：自定义Hook和封装面包屑

# 还是React-fam-bucketdemo项目，我们以商品更新为例学习

## 我们需要修改产品信息，但是首先需要获取分类的级联数据

## 新增商品参考图

|  |
| --- |
|  |

## 老师的接口

|  |
| --- |
|  |

## 我们把ProductsUpdate.jsx文件修改一下

|  |
| --- |
| **import *React***, {*useEffect*} **from 'react'**; **import** {*useLocation*, *useParams*} **from "react-router-dom"**; **import** {*findProductByIdApi*} **from "../../../api/products"**;  **const** *ProductsUpdate* = () => {  *//动态路由方式的接收参数  // let params = useParams();  // console.log(params)  //state传参方式接收参数* **let** location = *useLocation*()  ***console***.log(location)  **let** getProductById = **async** (id)=>{  **let** data = **await** *findProductByIdApi*(id)  ***console***.log(data)  }   *useEffect*(()=>{  *//动态路由的调用方式  // getProductById(params.id)  //state传参的调用方式* getProductById(location.**state**.**id**)  },[])  **return** (  <**div**>  update 页面  </**div**>  ); };  **export default** *ProductsUpdate*; |

## 1.我们打开src/api/category.js,添加一个查询所有分类的方法，注意：我的接口和老师的接口不一样，

### 我的接口

|  |
| --- |
| **import *request* from "../utils/request"**;  *//获取指定的分类* **export let** *getCategoryApi* = (params)=>***request***.get(**"/category"**,{params}) *//获取所有分类* **export let *getCascaderCategoryApi*** = ***request***.get(**"/category"**) |

### 老师的接口

|  |
| --- |
|  |

## 不知道怎么解决后端自动级联数据的方法，

# 课堂截图

## 1.创建一个状态cascaderData来保存数据

|  |
| --- |
|  |

## 2.在products文件夹下面新建一个新增组件ProductsAdd.jsx

|  |
| --- |
|  |

### 新增的表单界面是这样子的

|  |
| --- |
|  |

## 3.看看新增和修改有哪些功能是一样的

|  |
| --- |
|  |

## 4.另外，在新增组件里面也是需要使用useEffect的但是，调用有点不一样，需要自己写，不能复制

|  |
| --- |
|  |

## 5.新增和修改有很多代码是一样的，可不可以封装起来，方便使用？可以。我们在src文件下面新建应该hooks文件夹，在这个文件夹里面新建应该叫做useCascader.js的自定义hook文件，然后把公共代码抽取出来。

### 抽取公共代码

|  |
| --- |
|  |

### 放进src/hooks/useCascader.js并且做需要的优化处理(如果有的话)

|  |
| --- |
|  |

### 需要返回一些数据，可以根据需要导出

|  |
| --- |
|  |

### 然后在剔除了公共代码的组件里面使用我们的自定义hook，先看看能否拿到数据

|  |
| --- |
|  |

### 在ProductsUpdate组件里面也可以使用

|  |
| --- |
|  |

#### 效果

|  |
| --- |
|  |

## 我们也可以把UsersPage.jsx组件文件也进行自定义hook的提取，使用自定义hook的好处就是可以使得组件的代码比较简洁易懂。当组件的代码很长很复杂的时候，就需要使用自定义hook来优化代码。

# 现在学习面包屑封装

## 1.把IndexPage.jsx里面的面包屑代码单独提取出来，我们在src/components文件夹下面新建一个index-breadcrumb文件夹，然后在里面新建一个IndexBreadcrumb.jsx文件，把IndexPage.jsx里面的面包屑代码剪贴过来

|  |
| --- |
|  |
|  |

### 注意：Breadcrumb.Item已经过时了，需要使用新写法

## 2.在IndexPage.jsx中使用面包屑

|  |
| --- |
|  |

### 先保证基本功能不受影响

|  |
| --- |
|  |

## 3.在面包屑组件中把内容改为我们需要的内容，面包屑需要根据不同的路由显示不同的内容，而且是动态显示的。那怎么把路由和面包屑关联起来？我们可以定义一个数组，把路由和数据关联起来

|  |
| --- |
|  |

## 4.根据用户访问的路由实现面包屑的动态加载，。需要使用useLocation的第三方Hook

|  |
| --- |
|  |

## 注意：需要添加key，否则报错

|  |
| --- |
|  |

## 5.解决进入修改页面报错的问题，只需要在16行的value后面 加一个?和在21行的breadcrumbItem后面跟一个?

|  |
| --- |
|  |

### 效果如下

|  |
| --- |
|  |

### 不过还有有一个问题，修改页面也需要面包屑，否则给人一个界面不统一的效果

## 6.我们修改一些面包屑数据，添加修改商品的路由

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

## 不过此时有一个问题，就是使用动态路由 传参进入修改页面的时候

### ProductsList.jsx

|  |
| --- |
| **import *React***, {*useEffect*, *useState*} **from 'react'**; **import** { ***Divider***, ***Radio***, ***Table***,***Space*** } **from 'antd'**; **import** {*getProductsApi*} **from "../../../api/products"**; **import** {*Link*} **from "react-router-dom"**;  **const** columns = [  {  **title**: **'商品名称'**,  **dataIndex**: **'name'**,  render: (text) => <**a**>{text}</**a**>,  },  {  **title**: **'描述'**,  **dataIndex**: **'title'**,  },  {  **title**: **'价格'**,  **dataIndex**: **'price'**,  },  {  **title**: **'类别'**,  **dataIndex**: [**'type'**,**'name'**],  },  {  **title**: **'操作'**,  **dataIndex**: **'action'**,  render:(\_,record)=>{  **return** (  <**Space**>  {*/\*方式1 动态路由传参\*/*}  <**Link to=**{**"/products/update/"**+record.**id**}>修改</**Link**>  {*/\*方式2 state传参，还是需要Link\*/*}  <**Link to="/products/update" state=**{{**id**:record.**id**}}>修改</**Link**>  <**a**>删除</**a**>  </**Space**>  )  }  }, ];  *// rowSelection object indicates the need for row selection* **const** rowSelection = {  onChange: (selectedRowKeys, selectedRows) => {  ***console***.log(**`selectedRowKeys:** ${selectedRowKeys}**`**, **'selectedRows: '**, selectedRows);  } };  **const** *ProductsList* = () => {  *//状态* **let** [productsData,setProductsData] = *useState*([])  *//方法* **let** getProducts = **async** ()=>{  **let** data = **await** *getProductsApi*()  *// console.log(data)* setProductsData([...data])  ***console***.log(**"products:"**,productsData)  }   *useEffect*(()=>{  getProducts()   },[])  **return** (  <**div**>  <**Table  rowSelection=**{{**type**: **'checkbox'**, ...rowSelection}}  **columns=**{columns}  **dataSource=**{productsData}  **rowKey=**{**"id"**}  />  </**div**>  ); };  **export default** *ProductsList*; |

### App.jsx

|  |
| --- |
|  |

### 此时进入修改页面就没有面包屑了，因为此时的路由变了，就找不到对应的面包屑数据

## 7.把动态路由和面包屑关联起来，此时用pathname ==就匹配不上，此时就不能用==来判断，需要改为包含

|  |
| --- |
|  |

## 本来想用上面的方法，不过antd说它过时了，所以不要这么用，必须使用新方法，最好先进行数据封装，需要把获取到的breadcrumbsItem转化为一个对象数组，然后给Breadcrumb的items属性赋值

|  |
| --- |
| **import** {***Breadcrumb***} **from "antd"**; **import *React* from "react"**; **import** {*useLocation*} **from "react-router-dom"**; **import *BreadcrumbItem* from "antd/es/breadcrumb/BreadcrumbItem"**;  **let** breadcrumbsData = [  {**path**:**"/home"**,**value**:[**'首页'**]},  {**path**:**"/user"**,**value**:[**'首页'**,**'用户管理'**]},  {**path**:**"/roles"**,**value**:[**'首页'**,**'角色管理'**]},  {**path**:**"/products/category"**,**value**:[**'首页'**,**'商品管理'**,**'商品分类'**]},  {**path**:**"/products/list"**,**value**:[**'首页'**,**'商品管理'**,**'商品列表'**]},  {**path**:**"/products/update"**,**value**:[**'首页'**,**'商品管理'**,**'修改商品'**]}, ]  **const** *IndexBreadcrumb* = () => {  **let** {pathname} = *useLocation*()  **let** breadcrumbsItem = breadcrumbsData.find(item =>pathname.includes(item.**path**))?.**value** *// console.log(breadcrumbsItem)  //对上面的数据进行处理，返回一个类似于 {title:"Home"}的对象数组，然后在下面的items里面使用* **let** breadcrumbsItemobj = breadcrumbsItem.map(item=>{  **return** {  **title**:item  }  })  ***console***.log(breadcrumbsItemobj)  **return** (  <**Breadcrumb  style=**{{  **margin**: **'16px 0'**,  }}  **items=**{[  ...breadcrumbsItemobj  ]}  >  {*/\*{\*/*}  {*/\* breadcrumbsItem?.map((item,index) => (\*/*}  {*/\* <Breadcrumb.Item key={index}>{item}</Breadcrumb.Item>\*/*}  {*/\* ))\*/*}  {*/\*}\*/*}   </**Breadcrumb**>  ); };  **export default** *IndexBreadcrumb*; |