# React + typescript + antd开发前端应用（九）完整多页签应用基本框架

ep76 已关注

1 人赞同了该文章

上一篇：React + typescript + antd开发前端应用（八）使用全局状态

有了前面的基础知识，可以开始真正的多页签基本页面的开发了。antd的Tabs组件默认在页签体中显示的是一些简单的文本，因此需要Tabs组件、Outlet组件、react-activation及编程式导航等相互配合才能完成多页签应用框架页的开发。react-activation是类似vue的KeepAlive组件，主要提供页签切换时缓存以及加载的页签内容，都在每次激活页签，页面内容都会重新渲染一次。

在上一篇React + typescript + antd开发前端应用（八）使用全局状态文档中，我们完成了全局state的定义及使用的案例，同时还完成了多页签框架页的基本开发，本片文档将在此基础上完成tabs组件、react编程式路由导航及Outlet组件配合完成多页签开发。

## 1、创建首页及其他路由目标页面组件

1、创建src\demopages\RootPage.tsx文件

该文件作为首页页签的路由页面，文件内容如下：

```
import { Form, Input } from 'antd';
import React from 'react';
import { Link } from 'react-router-dom';
import { Outlet } from 'react-router-dom';

function RootPage() {
    return (
        <>
            <h2 style={{color: 'white', textAlign: 'center'}}>首页组件</h2>
            <Form autoComplete='off'>
                <Form.Item label='请输入地址：'><Input></Input></Form.Item>
            </Form>
        </>
    );
}

export default RootPage;
```

2、创建src\demopages\PageOne.tsx文件

该文件作为菜单项“功能一”对应页面，文件内容如下：

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

function PageOne() {
    return (
        <React.Fragment>
            <h2 style={{textAlign: 'center', color: 'white'}}>页面组件一</h2>
            <Form>
                <Form.Item l
            </Form>
```

```
        </React.Fragment>
    );
}

export default PageOne;
```

3、创建src\demopages\PageTwo.tsx文件

该文件作为菜单项“功能二”对应页面，文件内容如下：

```
import { Form, Input } from 'antd';
import React from 'react';
import { useLoaderData } from 'react-router-dom';

export async function loader(params: any) {
    console.log('loader');
    console.log(params)
    console.log(params.params.bizDataName);
    return {name: '查收数据库的数据'};
  }

function PageTow() {
    console.log('PageTow');
    const contact = useLoaderData();
    console.log(contact)
    return (
        <React.Fragment>
            <h1 style={{textAlign: 'center', color: 'white'}}>页面组件二</h1>
            <Form>
                <Form.Item label="地址" labelCol={{span: 2}} wrapperCol={{span: 3}}><I
            </Form>
        </React.Fragment>
    );
}

export default PageTow;
```

4、创建src\demopages\PageThree.tsx文件

该文件作为菜单项“功能三”对应页面，文件内容如下：

```
import React from 'react';
import { useLoaderData } from 'react-router-dom';

export async function loader(params: any) {
    console.log('loader');
    console.log(params)
    console.log(params.params.bizDataName);
    return {name: '查收数据库的数据'};
  }

function PageTow() {
    console.log('PageTow');
    const contact = useLoaderData();
    console.log(contact)
    return (
        <React.Fragment>
            <h1 style={{textAlign: 'center', color: 'white'}}>页面组件三</h1>
        </React.Fragment>
    );
```

```
}

export default PageTow;
```

## 2、定义路由文件

创建src\routers\RoutersDef.tsx文件，注意菜单功能页面对应组件的路由，必须是/（根）路由的子路由，文件内容如下：

```
import { createHashRouter } from 'react-router-dom';
import AppLayout from '../layout/AppLayout';
import RootPage from '../demopages/RootPage';
import PageOne from '../demopages/PageOne';
import PageTow from '../demopages/PageTwo';
import PageThree from '../demopages/PageThree';

const routes = createHashRouter([
    {
        path: '/',
        element: <AppLayout />,
        children: [
            {
                path: '/root',
                element: <RootPage />
            } , {
                path: '/one',
                element: <PageOne />
            } , {
                path: '/two',
                element: <PageTow />,
            } , {
                path: '/three',
                element: <PageThree />,
            }
        ]
    }
]);

export default routes;
```

## 3、修改src\index.tsx文件

主要是引入路由相关模块，并使用刚刚定义的路由数据，代码如下：

```
import React from 'react';
import { createRoot } from 'react-dom/client';
import 'antd/dist/reset.css';
import { RouterProvider } from 'react-router-dom';//引入路由模块
import routes from './routers/RoutersDef';//引入路由定义文件
import { AppContextProvider } from './context/AppContextProvider';

createRoot(document.getElementById('root') as HTMLElement).render(
  <React.StrictMode>
    <AppContextProvider>
      <RouterProvider router={ routes }></RouterProvider>{/** 添加路由支持 */}
    </AppContextProvider>
  </React.StrictMode>
);
```

## 4、修改src\layout\AppLayout.tsx文件

主要是引入react路由相关模块，代码如下：

```
import { useContext } from 'react';
import { Layout, ConfigProvider, theme } from "antd";
import './AppLayout.css';
import AppMenu from "./AppMenu";
import AppTabs from "./AppTabs";
import { AppContext } from '../context/AppContextProvider';
import { Outlet, useNavigate } from 'react-router-dom';//引入路由相关模块
import { useEffect } from 'react';

const { Header, Sider, Content } = Layout;

function AppLayout() {
    const {dataInfo, setDataInfo} = useContext(AppContext)!;
    const navigate = useNavigate();

    useEffect(() => {
        navigate('/root', {replace: true});//页面加载时设置首页页签显示内容
    }, []);

    function addNewTab(tabKey: string, tabLabel: string) {
        if (dataInfo.tabItems.some((oneItem) => oneItem.key === tabKey)) {
            console.log("change tab");
            let newDataInfo = {...dataInfo};
            newDataInfo.activeKey = tabKey;
            setDataInfo(newDataInfo);
        } else {
            console.log("change tab adddd");
            let newDataInfo = {...dataInfo};
            newDataInfo.activeKey = tabKey;
            newDataInfo.tabItems = [...dataInfo.tabItems, { label: tabLabel, children:
            setDataInfo(newDataInfo);

        }
        setTabPage(tabKey)
    };

    function selectedMenuKey(menuKeyId: string) {
        addNewTab(menuKeyId, '功能: ' + menuKeyId);
    }
    //根据选择的菜单项或标签页，显示和不同的路由对应的页面组件
    function setTabPage(tabKey: string) {
        if (tabKey === 'systemMenu_menu1') {
            navigate('/one', {replace: true});
        } else if (tabKey === 'systemMenu_menu2') {
            navigate('/two', {replace: true});
        } else if (tabKey === 'systemMenu_menu3') {
            navigate('/three', {replace: true});
        } else {
            navigate('/root', {replace: true});
        }
    }
    return (
        <ConfigProvider theme={{ algorithm: theme.darkAlgorithm }}>
            <Layout>
                <Header>Header部分</Header>
                <Layout>
                    <Sider width={190} style={{overflow: "auto"}}>
                        <Appl
                    </Sider>
```

```
                    <Content>
                        <AppTabs selectedMenuKey={selectedMenuKey}/>
                        <Outlet />{/** 作为路由组件的渲染出口 */}
                    </Content>
                </Layout>
            </Layout>
        </ConfigProvider>
    );
};

export default AppLayout;
```

## 5、修改src\layout\AppTabs.tsx文件

选择标签页时调用父组件的方法设置当前路由，让Outlet显示对应的路由组件，达到多页签的效果：

```
import {useContext} from 'react';
import { Tabs } from 'antd';
import { TargetKey } from './AppTabsFuncs';
import { AppContext } from '../context/AppContextProvider';

function AppTabs(props: {selectedMenuKey: (tabKeyId: string) => void}) {
    const {dataInfo, setDataInfo} = useContext(AppContext)!;
    function onChange (newActiveKey: string) {
        let newDataInfo = {...dataInfo};
        newDataInfo.activeKey = newActiveKey;
        setDataInfo(newDataInfo);
        props.selectedMenuKey(newActiveKey);//调用父组件函数，设置页签对应路由
    };
    function remove(targetKey: TargetKey) {
        let newActiveKey = dataInfo.activeKey;
        let lastIndex = -1;
        dataInfo.tabItems.forEach((item, i) => {
            if (item.key === targetKey) {
                lastIndex = i - 1;
            }
        });
        const newPanes = dataInfo.tabItems.filter((item) => item.key !== targetKey);
        if (newPanes.length && newActiveKey === targetKey) {
            if (lastIndex >= 0) {
                newActiveKey = newPanes[lastIndex].key;
            } else {
                newActiveKey = newPanes[0].key;
            }
        }
        let newState = {...dataInfo};
        newState.activeKey = newActiveKey;
        newState.tabItems = newPanes;
        setDataInfo(newState);
    };
    function onEdit(targetKey: React.MouseEvent | React.KeyboardEvent | string, action
        if (action === 'remove') {
            remove(targetKey);
        }
    };
    return (
        <Tabs
            type="editable-card"
            onChange={(newKe
```

```
            activeKey={dataInfo.activeKey}
            onEdit={(targetKey, action) => {onEdit(targetKey, action)}}
            items={dataInfo.tabItems}
            hideAdd={true}
            defaultActiveKey='1'
        />
    );
};

export default AppTabs;
```

## 6、解决Outlet组件有Tabpane的文本内容的问题

增加样式，隐藏Tabpane内容即可。修改src\layout\AppLayout.css文件：

```
#root, main, .ant-layout {
    height: 100%;
}
.ant-tabs-content-holder {
    display: none; /* 因此Tabpane内容 */
}
```

## 7、解决tab页切换时重新加载组件的问题

完成以上工作后，经测试发现存在以下问题：在tab1中的输入框输入内容后，切换tabs到tab2页签，再次点击tab1切换会tab1后，输入的内容丢失了。实际多页签效果应该时：页签切换时，需要保留原来的页签状态，关闭页签后，点击菜单重新创建页签时需要重新初始化整个页签内容。要解决整个问题，需要用到一个react插件：react-activation。在使用react-activation前，需要安装该组件，在工程目录执行以下命令：

```
npm i react-activation --save
```

1、修改src\index.tsx取消严格模式

```
import { createRoot } from 'react-dom/client';
import 'antd/dist/reset.css';
import { RouterProvider } from 'react-router-dom';
import routes from './routers/RoutersDef';
import { AppContextProvider } from './context/AppContextProvider';

createRoot(document.getElementById('root') as HTMLElement).render(
  <AppContextProvider>
    <RouterProvider router={routes}></RouterProvider>
  </AppContextProvider>
);
```

2、修改src\routers\RoutersDef.tsx文件

为路由增加组件<KeepAlive>，代码如下：

```
import { createHashRouter } from 'react-router-dom';
import AppLayout from '../layout/AppLayout';
import RootPage from '../demopages/RootPage';
import PageOne from '../demopages/PageOne';
import PageTow from '../demo
import PageThree from '../de
```

```
import KeepAlive from 'react-activation';

const routes = createHashRouter([
    {
        path: '/',
        element: <AppLayout />,
        children: [
            {
                path: '/root',
                //增加KeepAlive组件，并为组件指定name和cacheKey，注意每个cacheKey不能相同
                element: <KeepAlive name='root' cacheKey="UNIQUE_ID1"><RootPage /></Ke
            } , {
                path: '/one',
                element: <KeepAlive name='systemMenu_menu1' cacheKey="UNIQUE_ID2"><Pag
            } , {
                path: '/two',
                element: <KeepAlive name='systemMenu_menu2' cacheKey="UNIQUE_ID3"><Pag
            } , {
                path: '/three',
                element: <KeepAlive name='systemMenu_menu3' cacheKey="UNIQUE_ID4"><Pag
            }
        ]
    }
]);

export default routes;
```

3、修改src\layout\AppLayout.tsx文件

用<AliveScope>组件包裹AppLayout组件，代码如下：

```
import { useContext } from 'react';
import { Layout, ConfigProvider, theme } from "antd";
import './AppLayout.css';
import AppMenu from "./AppMenu";
import AppTabs from "./AppTabs";
import { AppContext } from '../context/AppContextProvider';
import { Outlet, useNavigate } from 'react-router-dom';
import { useEffect } from 'react';
import { AliveScope } from 'react-activation';

const { Header, Sider, Content } = Layout;

function AppLayout() {
    const {dataInfo, setDataInfo} = useContext(AppContext)!;
    const navigate = useNavigate();

    useEffect(() => {
        navigate('/root', {replace: true});//页面加载时设置首页页签显示内容
    }, []);

    function addNewTab(tabKey: string, tabLabel: string) {
        if (dataInfo.tabItems.some((oneItem) => oneItem.key === tabKey)) {
            console.log("change tab");
            let newDataInfo = {...dataInfo};
            newDataInfo.activeKey = tabKey;
            setDataInfo(newDataInfo);
        } else {
            console.log("change tab adddd");
            let newDataInfo = {...dataInfo};
            newDataInfo.acti
            newDataInfo.tabI
```

```
            setDataInfo(newDataInfo);

        }
        setTabPage(tabKey)
    };

    function selectedMenuKey(menuKeyId: string) {
        addNewTab(menuKeyId, '功能：' + menuKeyId);
    }

    function setTabPage(tabKey: string) {
        if (tabKey === 'systemMenu_menu1') {
            navigate('/one', {replace: true});
        } else if (tabKey === 'systemMenu_menu2') {
            navigate('/two', {replace: true});
        } else if (tabKey === 'systemMenu_menu3') {
            navigate('/three', {replace: true});
        } else {
            navigate('/root', {replace: true});
        }
    }
    return (
        <ConfigProvider theme={{ algorithm: theme.darkAlgorithm }}>
            <AliveScope>{/** 增加的组件 */}
                <Layout>
                    <Header>Header部分</Header>
                    <Layout>
                        <Sider width={190} style={{overflow: "auto"}}>
                            <AppMenu selectedMenuKey={selectedMenuKey} />
                        </Sider>
                        <Content>
                            <AppTabs selectedMenuKey={selectedMenuKey} setTabPage={set
                            <Outlet />{/** 作为路由组件的渲染出口 */}
                        </Content>
                    </Layout>
                </Layout>
            </AliveScope>
        </ConfigProvider>
    );
};

export default AppLayout;
```

4、修改src\layout\AppTabs.tsx文件

在tab页签关闭时手工销毁组件缓存，代码如下：

```
import {useContext} from 'react';
import { Tabs } from 'antd';
import { TargetKey } from './AppTabsFuncs';
import { AppContext } from '../context/AppContextProvider';
import { useAliveController } from 'react-activation';

function AppTabs(props: {selectedMenuKey: (tabKeyId: string) => void, setTabPage: (tab
    const {dropScope} = useAliveController()
    const {dataInfo, setDataInfo} = useContext(AppContext)!;
    function onChange (newActiveKey: string) {
        let newDataInfo = {...dataInfo};
        newDataInfo.activeKey = newActiveKey;
        setDataInfo(newDataInfo);
        props.selectedMenuKe
    };
```

```
    function remove(targetKey: TargetKey) {
        let newActiveKey = dataInfo.activeKey;
        let lastIndex = -1;
        dataInfo.tabItems.forEach((item, i) => {
            if (item.key === targetKey) {
                lastIndex = i - 1;
            }
        });
        const newPanes = dataInfo.tabItems.filter((item) => item.key !== targetKey);
        if (newPanes.length && newActiveKey === targetKey) {
            if (lastIndex >= 0) {
                newActiveKey = newPanes[lastIndex].key;
            } else {
                newActiveKey = newPanes[0].key;
            }
        }
        let newState = {...dataInfo};
        newState.activeKey = newActiveKey;
        newState.tabItems = newPanes;
        setDataInfo(newState);
        dropScope(targetKey.toString());//删除页签时，根据路由中定义的name，删除缓存
        props.setTabPage(newState.activeKey);
    };
    function onEdit(targetKey: React.MouseEvent | React.KeyboardEvent | string, action
        if (action === 'remove') {
            remove(targetKey);
        }
    };
    return (
        <Tabs
            type="editable-card"
            onChange={(newKeyId) => {onChange(newKeyId) }}
            activeKey={dataInfo.activeKey}
            onEdit={(targetKey, action) => {onEdit(targetKey, action)}}
            items={dataInfo.tabItems}
            hideAdd={true}
            defaultActiveKey='1'
        />
    );
};

export default AppTabs;
```

调整代码完成后，就可以解决切换页签时未缓存组件的问题了。

react-activation官网说：(React v18+) 不要使用 `ReactDOMClient.createRoot`，而是使用 `ReactDOM.render`，但我测试使用 ReactDOMClient.createRoot好像没问题，待后续实际工作中验证是否有问题把。

## 8、react项目国际化

react国际化依赖react-intl插件支持，需要执行以下命令安装react-init：

```
npm i react-intl @types/react-intl --save
```

1、在创建src\locales目录，在该目录下创建多语言资源文件，如en_US.ts文件内容如下：

```
const en_US = {
    hello: 'Hello world'
```

```
};

export default en_US;
```

创建zh_CN.ts文件内容如下：

```
const zh_CN = {
    hello: '你好世界'
};

export default zh_CN;
```

创建index.ts文件，内容如下：

```
import zh_CN from './zh_CN';
import en_US from './en_US';
import zhCN from 'antd/locale/zh_CN';
import enUS from 'antd/locale/en_US';

function loadLocales(lang: string) {
    let locale = null;
    let message = null;
    let antLocale = null;
    switch (lang) {
        case 'en_US':
            locale = 'en-US';
            message = en_US;
            antLocale = enUS;
            break;
        case 'zh_CN':
            locale = 'zh-CN';
            message = zh_CN;
            antLocale = zhCN;
            break;
        default:
            locale = 'zh-CN';
            message = zh_CN;
            antLocale = zhCN;
            break;
    }
    return { locale, message, antLocale };
}
export { loadLocales };
```

2、修改AppLayout.tsx文件，将需要国际化的资源包裹起来，代码如下：

```
const {locale, message, antLocale} = loadLocales('zh_CN');//需要显示的语言内容，可以通过A
……
return (
    <ConfigProvider theme={{ algorithm: theme.darkAlgorithm }} locale={antLocale}>
        <IntlProvider messages={message} locale={locale}>{/** 增加国际化支持 */}
            <AliveScope>
                <Layout>
                    <Header>Header部分</Header>
                    <Layout>
                        <Sider width={190} style={{overflow: "auto"}}>
                            <AppMenu selectedMenuKey={selectedMenuKey} />
                        </Sider>
                        <Content>
```

```
                    <Outlet />{/** 作为路由组件的渲染出口 */}
                </Content>
            </Layout>
        </Layout>
    </AliveScope>
</IntlProvider>
</ConfigProvider>
);
```

3、使用组件或api读取语言资源信息：

```
return (
    <>
        <h2 style={{color: 'white', textAlign: 'center'}}>首页组件</h2>
        <Form autoComplete='off'>
            <Form.Item label='请输入地址：'><Input></Input></Form.Item>
            <Form.Item label='请选择日期：'><DatePicker></DatePicker></Form.Item>
            <FormattedMessage id='hello'/>{/** 语言返回不同的内容 */}
            <Button>{intl.formatMessage({id: 'hello'})}</Button><br />{/** 语言返回不同
            <Button>改变语言</Button>
        </Form>
    </>
);
```

至此，React应用国际化至此完成。代码const {locale, message, antLocale} = loadLocales('zh_CN')中的语言配置可以通过AppContext获取全局配置，然后利用useEffect钩子函数在，在AppContext中的语言发生变化时翻译页面内容为自动的语言信息。

## 9、本系列文档所使用的相关操作系统命令

```
npm i create-react-app -g
npm config set registry https://registry.npmmirror.com
create-react-app mydemopro --template typescript
cd mydemopro
npm i antd -S
npm i react-router-dom -S
npm i @types/react-router-dom -S
npm i react-activation -S
npm i react-intl @types/react-intl -S
```

编辑于 2024-03-16 01:42 · IP 属地上海

Angular　Ant Design　React

欢迎参与讨论

**2 条评论**　默认　最新

**mini**

大佬,这种打开太多,卡顿问题怎么解决吗

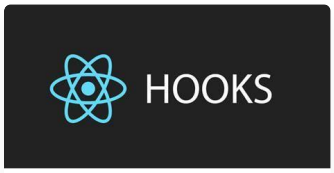
2023-12-03 · IP 属地浙江　回复　喜欢

**ep76** 作者

一般在应用中都限制打

2023-12-03 · IP 属地上海 回复 喜欢

**推荐阅读**




**《reactHook+Typescript 从入门到实践》年底让这篇文章...**

Ruoduan

**TypeScript中高级应用与最佳实践**

原文：TypeScript 中高级应用与最佳实践 作者：TAT.haoyueTypeScript的定位JavaScript的超集编译期行为不引入额外开销不改变运行时行为始终与ESMAScript 语言标准一致 (stag...

Georg... 发表于Alloy...

**你不知道的 TypeScript 泛型（万字长文，建议收藏）**

泛型是 TypeScript（以下简称 TS）比较高级的功能之一，理解起来也比较困难。泛型应用场景非常广泛，很多地方都能看到它的影子。平时我们阅读开源 TS 项目源码，或者在自己的 TS 项目中使...

lucif... 发表于力扣加加

**类型门到**

一只