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

ep76 已关注

上一篇：React + typescript + antd开发前端应用（七）添加菜单点击功能

对于一些需要频繁跨组件访问的state，如果采用在父组件定义state，然后通过组件的组件属性向下属组件传递的方式来访问，一旦组件的层级较深，那这种方式将是一场灾难。另外，为菜单添加点击事件的代码来看，对于更新state的函数被传递到子组件以及行为组件中的方式并不科学，对代码的可读性造成了困扰，而且在各个组件间传递的参数较多，很容易产生Bug。在后续的代码中，还是根据情况，不必追求将UI组件和行为组件完全分离的模式。

## 1、创建AppContext及AppContextProvider

分析之前的代码，tabs组件需要两个state：一个是记录当前激活的标签页的key，另一个是tabs页签项的数组对象。这两个state可能需要在AppTabs组件和AppMenu中访问，因此需要创建一个全局对象来保存这两类数据。创建src\context\AppContextProvider.tsx文件，编辑内容如下：

```
import { createContext, useState } from 'react';
import { TabItem } from '../layout/AppLayoutFuncs';//文件内容由修改，在后续代码端说明
//定义state的数据类型
type StateType = {
    activeKey: string,//当前处于激活状态的tabitem的key
    tabItems: TabItem[]//tabs的tab页签对象数组
}
//定义全局state对象及更新全局state的函数
type ComplexObject = {
    dataInfo: StateType,
    setDataInfo: (dataInfo: StateType) => void,
};
//根据定义创建Context
export const AppContext = createContext<ComplexObject | null>(null);
//定义ContextProvider，children是组件
export function AppContextProvider(props: {children: React.ReactNode | React.ReactNode
    //调用useState创建state
    const [dataInfo, setDataInfo] = useState<StateType>({ activeKey: '1' ,
        tabItems: [{ label: '首页', children: '首页页签内容', key: '1', closable: false
    return (
        <AppContext.Provider value={{dataInfo, setDataInfo}}>{/** value就是可在<AppCont
            {props.children}
        </AppContext.Provider>
    );
}
```

## 2、修改index.tsx

主要用刚刚传教的AppContextProvider包裹AppLayout组件，以便在AppLayout及其子组件中使用全局state对象及函数：

```
import React from 'react';
import { createRoot } from 'react-dom/client';
import 'antd/dist/reset.css';
import { AppContextProvider
import AppLayout from './lay
```

```
createRoot(document.getElementById('root') as HTMLElement).render(
  <React.StrictMode>
    <AppContextProvider>{/** 添加AppContextProvider组件，包裹组件 */}
      <AppLayout />
    </AppContextProvider>
  </React.StrictMode>
);
```

### 3、修改src\layout\AppLayoutFuncs.ts文件

文件移除了行为函数，否则state数据、更新state的函数都需要作为参数传递到函数中，对代码的可读性造成了困扰：

```
//定义Tabs的每个标签页对象的数据结构
export type TabItem = { label: string, children: string, key: string, closable?: boole
```

### 4、修改src\layout\AppLayout.tsx文件

使用AppContext，添加行为函数：

```
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';//导入AppContext

const { Header, Sider, Content } = Layout;

function AppLayout() {
    const {dataInfo, setDataInfo} = useContext(AppContext)!;//使用AppContext，注意函数后
    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);
        }
    };
    function selectedMenuKey(menuKeyId: string) {
        addNewTab(menuKeyId, '功能: ' + menuKeyId);
    }
    return (
        <ConfigProvider theme={{ algorithm: theme.darkAlgorithm }}>
            <Layout>
                <Header>Header部分</Header>
                <Layout>
                    <Sider width={190} style={{overflow: "auto"}}>
                        <AppMenu selectedMenuKey={selectedMenuKey} />
                    </Sider>
                    <Content
```

```
                        <AppTabs />{/** 不在需要传递state以及更新state的函数等参数 */}
                    </Content>
                </Layout>
            </Layout>
        </ConfigProvider>
    );
};

export default AppLayout;
```

## 5、修改src\layout\AppTabsFuncs.ts文件

删除行为函数：

```
//标签页点击事件对象
export type TargetKey = React.MouseEvent | React.KeyboardEvent | string;
```

## 6、修改src\layout\AppTabs.tsx文件

使用全局state：

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

function AppTabs() {
    const {dataInfo, setDataInfo} = useContext(AppContext)!;//所以用AppContext
    //点击某页签处理逻辑：设置页签为激活状态
    function onChange (newActiveKey: string) {
        let newDataInfo = {...dataInfo};
        newDataInfo.activeKey = newActiveKey;
        setDataInfo(newDataInfo);
    };
    //点击页签右上角X关闭页签是处理逻辑：1、删除当前页签，2、设置当前页签的前要给页签为激活状态
    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(targetKe: React.MouseEvent | React.KeyboardEvent | string, action
        if (action === 'remo
```

```
            remove(targetKey);
        }
    };
    return (
        <Tabs
            type="editable-card"
            onChange={(newKeyId) => {onChange(newKeyId) }}
            activeKey={dataInfo.activeKey}
            onEdit={(targetKey, action) => {onEdit(targetKey, action)}}
            items={dataInfo.tabItems}{/** 使用全局的页签对象数组 */}
            hideAdd={true}
            defaultActiveKey='1'
        />
    );
};

export default AppTabs;
```

## 7、修改src\layout\AppMenuFuncs.ts文件

```
import type { MenuProps } from 'antd';

//定义每个菜单项的数据对象
export type MenuItem = Required<MenuProps>['items'][number];
/**
 * 根据参数生成菜单项对象
 * @param label 菜单项目显示文字
 * @param key 菜单项唯一key
 * @param icon 菜单项图标
 * @param children 子菜单数组
 * @param type 菜单类型，由子菜单是值为group
 * @returns 返回菜单项实例对象
 */
export function getItem(label: React.ReactNode, key: React.Key, icon?: React.ReactNode
    //等同于：return {key: key, icon: icon, children: children, label: label, type: typ
    return {key, icon, children, label, type} as MenuItem;
}
//菜单初始化数据数组
export const menuItems: MenuProps['items'] = [
    getItem('系统菜单', 'systemMenu', null, [
        getItem('功能一', 'systemMenu_menu1'),
        getItem('功能二', 'systemMenu_menu2'),
        getItem('功能三', 'systemMenu_menu3')
    ])
];
//<AppMenu>组件所需父组件传递的数据类型
export type Param = {
    selectedMenuKey: (keyId: string) => void
};
```

## 8、修改src\layout\AppMenu.tsx文件

```
import type { MenuProps } from 'antd';
import { Menu } from 'antd';
import { Param } from './AppMenuFuncs';
import { menuItems } from './AppMenuFuncs';

function AppMenu(props: Para
```

```
    //选中某菜单时添加页签并选中，调用父组件定义的selectedMenuKey函数
    const onClick: MenuProps['onSelect'] = (e) => {
        props.selectedMenuKey(e.key);
    };
    return (
        <Menu onSelect={onClick} defaultOpenKeys={['systemMenu']} mode="inline" items=
    );
}

export default AppMenu;
```

完成代码重构后，再次回到浏览器中进行测试，验证全局state是否能够正常工作。

依然建议读者不要copy这些代码，而是一行一行的编写这些代码，在编写过程中立即每行代码存在的意义，这对熟悉React和typescript都有好处。

下一篇：React + typescript + antd开发前端应用（九）完整多页签应用基本框架

编辑于 2023-10-13 10:00 · IP 属地上海

React Angular Ant Design

欢迎参与讨论

还没有评论，发表第一个评论吧

**推荐阅读**

**TypeScript在react项目中的实践**

慕课网 发表于猿论

**Electron + TypeScript + React 开发问题及技巧整理**

背景最近在开发一款内部工具软件，为了保持技术栈的统一性，以及为日后的跨平台支持考虑，选择了 Electron + TypeScript + React 的工程架构。这三者我接触的都不太多，所以从刚开始上手就...

b1gm0... 发表于开发杂录


**2021 年 TypeScript + React 工程化指南**

Henry 发表于阿里 CC...

**我是 Typ**

Vorte