环境

```
node:
16.17.1

npm:
8.15.0
```

Ant Design of React官网：https://ant.design/docs/react/introduce-cn

一、侧边栏的收缩和展开做调整

页面一刷新，默认是选中的是第一个栏目，Home.tsx页面中，默认选中的key数组更换一下：

```
<Menu theme="dark" defaultSelectedKeys={['/page1']} mode="inline" items={items}
onClick={menuClick} />
```

想要完成点击展开新一项的时候其他的收起来，在Home.tsx页面的Menu组件补充这1个属性 和1个事件：

```
const View: React.FC = () => {
    ...
    // openKeys 当前所有展开着的数组，一开始给空默认不展示
    const [openKeys, setOpenKeys] = useState(['']);
    const hdOpenChange = (keys:string[])=>{
        // 点击展开和收缩的时候执行这里的代码
        // console.log(keys);  // 这些展开这的keys的数组
        // 设置为最后一项
        setOpenKeys([keys[keys.length-1]]);
    }


    return (
       ...
            <Menu theme="dark" defaultSelectedKeys={['/page1']} mode="inline"
items={items} onClick={menuClick}
                // 补充这1个属性 和1个事件：
                // openKeys 表示当前所有展开着的数组
            openKeys={openKeys}
                // 点击展开和收缩的时候执行这里的代码
            onOpenChange={hdOpenChange}/>
    )
}
```

二、主菜单组件的抽取

Menu组件有他独立的业务逻辑，需要处理成子组件。

在components中新建MainMenu文件夹，在其中新建index.tsx，把Home.tsx中的Menu部分抽取过来：

```
import {
    DesktopOutlined,
    FileOutlined,
    PieChartOutlined,
    TeamOutlined,
    UserOutlined,
  } from '@ant-design/icons';
import React, { useState } from 'react';
import type { MenuProps } from 'antd';
import { Menu } from 'antd';
import { useNavigate } from "react-router-dom"
type MenuItem = Required<MenuProps>['items'][number];

function getItem(
  label: React.ReactNode,
  key: React.Key,
  icon?: React.ReactNode,
  children?: MenuItem[],
): MenuItem {
  return {
    key,
    icon,
    children,
    label,
  } as MenuItem;
}

const items: MenuItem[] = [
  getItem('栏目 1', '/page1', <PieChartOutlined />),
  getItem('栏目 2', '/page2', <DesktopOutlined />),
  getItem('栏目 3', 'sub1', <UserOutlined />, [
    getItem('Tom', '3'),
    getItem('Bill', '4'),
    getItem('Alex', '5'),
  ]),
  getItem('Team', 'sub2', <TeamOutlined />, [getItem('Team 1', '6'),
getItem('Team 2', '8')]),
  getItem('Files', '9', <FileOutlined />),
];
const mainMenu: React.FC = () => {

    const navigateTo = useNavigate()

  const menuClick = (e:{key:string})=>{
    console.log("点击了菜单", e.key);

    // 点击跳转到对应的路由    编程式导航跳转，  利用到一个hook
    navigateTo(e.key);
  }

  // openKeys 当前所有展开着的数组，一开始给空默认不展示
  const [openKeys, setOpenKeys] = useState(['']);
  const hdOpenChange = (keys:string[])=>{
    // 点击展开和收缩的时候执行这里的代码
    // console.log(keys);  // 这些展开这的keys的数组
    setOpenKeys([keys[keys.length-1]]);
  }
    return (
```

```
        <Menu theme="dark" defaultSelectedKeys={['/page1']} mode="inline" items=
{items} onClick={menuClick}
        openKeys={openKeys}
        onOpenChange={hdOpenChange}/>
    )
}

export default mainMenu
```

Home.tsx剩下的代码如下：

```
import { Breadcrumb, Layout } from 'antd';
import React, { useState } from 'react';
import { Outlet,useNavigate } from "react-router-dom"
import MainMenu from "@/components/MainMenu"
const { Header, Content, Footer, Sider } = Layout;


const View: React.FC = () => {
  const [collapsed, setCollapsed] = useState(false);

  return (
    <Layout style={{ minHeight: '100vh' }}>
      {/* 左边侧边栏 */}
      <Sider collapsible collapsed={collapsed}>
        <div className="logo"></div>
        {/* 主菜单*/}
        <MainMenu></MainMenu>
      </Sider>
      {/* 右边内容 */}
      <Layout className="site-layout">
        {/* 右边头部 */}
        <Header className="site-layout-background" style={{ paddingLeft: '16px'
}} >
          {/* 面包屑 */}
          <Breadcrumb style={{ lineHeight:'64px' }}>
              <Breadcrumb.Item>User</Breadcrumb.Item>
              <Breadcrumb.Item>Bill</Breadcrumb.Item>
            </Breadcrumb>
        </Header>
        {/* 右边内容部分-白色底盒子 */}
        <Content style={{ margin: '16px 16px 0' }} className="site-layout-
background">
            {/* 窗口部分 */}
            <Outlet />
        </Content>
        {/* 右边底部 */}
        <Footer style={{ textAlign: 'center', padding:0, lineHeight:"48px"
}}>Ant Design ©2018 Created by Ant UED</Footer>
      </Layout>
    </Layout>
  );
};

export default View;
```

三、菜单数据的整理

MainMenu组件中，getItem函数和 items 数组可以换成以下items数组：

```
// 登录请求到数据之后就可以跟items这个数组匹配了
const items: MenuItem[] = [
   {
    label: '栏目 1',
    key: '/page1',
    icon: <PieChartOutlined />,
   },
   {
    label: '栏目 2',
    key: '/page2',
    icon: <DesktopOutlined />,
   },
   {
    label: '栏目 3',
    key: 'page3',
    icon: <UserOutlined />,
    children:[
        {
            label: '栏目 301',
            key: '/page3/page301',
        },
        {
            label: '栏目 302',
            key: '/page3/page302',
        },
        {
            label: '栏目 303',
            key: '/page3/page303',
        }
    ]
   },
   {
    label: '栏目 4',
    key: 'page4',
    icon: <TeamOutlined />,
    children:[
        {
            label: 'Team 1',
            key: '/page4/page401',
        },
        {
            label: 'Team 2',
            key: '/page4/page402',
        }
    ]
   },
   {
    label: '栏目 5',
    key: 'page5',
    icon: <FileOutlined />,
   }
];
```

四、配置其余路径的跳转

router/index.tsx中：

```
{
    path:"/",
    element: <Home />,
    children:[
      {
        path:"/page1",
        element: withLoadingComponent(<Page1 />)
      },
      {
        path:"/page2",
        element: withLoadingComponent(<Page2 />)
      },
      {
        path:"/page3/page301",
        element: withLoadingComponent(<Page301 />),
      }
    ]
  },
  //  嵌套路由 结束--------------------
  // 访问其余的都直接到首页：
  {
    path:"*",
    element:<Navigate to="/page1"/>
  },
```

访问http://localhost:3002/page3/page301，发现当前样式没有跟上。样式是由菜单组件的defaultSelectedKeys属性决定的。

MainMenu.tsx中：

```
import { useNavigate,useLocation } from "react-router-dom"
...

const mainMenu: React.FC = () => {

    const navigateTo = useNavigate()

    const currentRoute =  useLocation()
    console.log("加载了菜单",currentRoute); //如果发现加载两次，这是开发环境下才会生产环境就不会了，在main.tsx把严格，模式标签去掉就不会了。至于为什么react要它加载两次详情见：https://blog.csdn.net/HYHhmbb/article/details/125973790

    ...
    ...
return (
        // defaultSelectedKeys={[currentRoute.pathname]} 处理当前项的样式
        // openKeys={openKeys} 展开哪些项
        <Menu theme="dark" defaultSelectedKeys={[currentRoute.pathname]} mode="inline" items={items} onClick={menuClick}
        openKeys={openKeys}
        onOpenChange={hdOpenChange}/>
    )
```

五、处理进入到页面时候，需要展开的项目

MainMenu.tsx中：

```
    // 处理当前需要展开的项目
    let firstOpenKeys:string = ""
    function findKey(obj:{key:string}) {
        return obj.key === currentRoute.pathname;
    }
    for(let i =0;i<items.length;i++){
        if(items[i]!['children'] && items[i]!['children'].length>1&&items[i]!
['children'].find(findKey)){
            console.log(items[i]!.key);
            firstOpenKeys = items[i]!.key as string;
            break;
        }
    }
```

解决items[i]!['children']的红色曲线警告：

我们在项目中的tsconfig.json文件中添加：`"suppressImplicitAnyIndexErrors":true,`选项，**重启Vscode**！

详细查看官方文档：https://www.typescriptlang.org/tsconfig#suppressImplicitAnyIndexErrors

```
"compilerOptions": {
    "suppressImplicitAnyIndexErrors":true,
    ...
}
```

六、登录页面

6.1、登录页面基本配置

/src/views/下创建以下目录:

```
|---views
    |---Login
        |---index.tsx
        |---init.ts
        |---login.module.scss
```

/src/views/Login/index.tsx

```
import { useEffect } from 'react'
import initLoginBg from "./init.ts"
import styles from "./login.module.scss"


const View = ()=> {

    // 背景处理
    useEffect(()=>{
```

```
        initLoginBg()
        window.onresize = function(){initLoginBg()}
    },[]);

    return (
    <div className={styles.loginPage}>
        <canvas id="canvas" style={{display:"block"}}></canvas>
        <div className={styles.loginBox}>
            <div className={styles.title}>
                <h1>前端乐哥 · 通用后台系统</h1>
                <p>Strive Everyday</p>
            </div>
        </div>
    </div>
    )
}

export default View
```

login.module.scss

```
.loginPage{
    position: relative;
    .loginBox{
        width: 450px;
        position: absolute;
        left: 50%;
        top: 50%;
        transform: translate(-50%,-50%);
        color:#fff;

        h1{
            font-weight: bold;
            font-size: 22px;
            text-align: center;
            color:#fff;
        }
        p{
            text-align: center;
            margin: 20px 0;
        }
        .title{
            margin-bottom: 40px;
            position: relative;
            &:before,&:after{
                content:"";
                width: 100px;
                height: 2px;
                position: absolute;
                background: linear-gradient(to
right,rgba(255,255,255,0),#1976D2);
                left: -20px;
                top:18px;
            }
            &:after{
                left: auto;
```

```
                background: linear-gradient(to
left,rgba(255,255,255,0),#1976D2);
                right: -20px;
            }
        }
    }
}
```

init.ts的引入报红色曲线警告，需要在src/vite-env.d.ts这个文件中声明ts模块

```
declare module '*.ts'
```

6.2、首页的表单绘制

找到Antd中的表单组件文档：https://ant.design/components/input-cn/

/src/views/Login/index.tsx中：

```
import { useEffect } from 'react'
import { Button, Input, Space } from 'antd';
import initLoginBg from "./init.ts"
import styles from "./login.module.scss"
import "./login.less"
import 'antd/dist/antd.css';

const View = ()=> {

    // 背景处理
    useEffect(()=>{
        initLoginBg()
        window.onresize = function(){initLoginBg()}
    },[]);


    return (
    <div className={styles.loginPage}>
        <canvas id="canvas" style={{display:"block"}}></canvas>
         {/* 手动添加固定类名    className={styles.loginBox+" loginbox"} */}
        <div className={styles.loginBox+" loginbox"}>
            <div className={styles.title}>
                <h1>前端乐哥 · 通用后台系统</h1>
                <p>Strive Everyday</p>
            </div>

            <div className='from'>
            <Space direction="vertical" size="large" style={{ display: 'flex'
}}>
                <Input  placeholder="用户名" className='ipt'/>
                <Input.Password  placeholder="密码" className='ipt'/>
                <Button  type='primary' block>登录</Button>
            </Space>
            </div>
        </div>
    </div>
    )
}
```

```
export default View
```

Login目录下新建login.less

```
// 登录页表单样式
.loginbox {
    // 控制表单元素
    .ant-input, .ant-input-password{
        background-color: rgba(255,255,255,0);
        border-color: #40a9ff;
        color:#fff;
    }
    // 控制眼睛图标
    .ant-input-password-icon.anticon,.ant-input-password-icon.anticon:hover{
        color:#40a9ff;
    }
}
```

6.3、验证码布局

/src/views/Login/index.tsx中：

```
<div className='captchaBox'>
    <Input  placeholder="验证码" className='ipt'/>
    <div className="captchaImg">
    <img src="xxx" height="38" alt="" />
    </div>
</div>
```

/src/views/Login/login.less中：

```
.loginbox {
     // 控制表单元素
    .ant-input, .ant-input-password{
        background-color: rgba(255,255,255,0);
        border-color: #40a9ff;
        color:#fff;
        height: 38px;
    }
    // 控制眼睛图标
    .ant-input-password-icon.anticon,.ant-input-password-icon.anticon:hover{
        color:#40a9ff;
    }
    // 控制placeholder文字样式
    .ant-input::-webkit-input-placeholder {
        color: rgba(64,169,255,0.6);
    }
    // 单独控制密码高度
    .ant-input-password .ant-input{
        height: 28px;
    }
    // 控制验证码盒子
    .captchaBox{
        display: flex;
```

```
        .ant-input{
            flex:1;
        }
        .captchaImg{
            cursor: pointer;
            margin-left: 20px;
        }
    }
    // 控制登录按钮
    .loginBtn{
        height: 38px;
    }
}
```

6.4、点击登录按钮获取用户输入信息

Login.tsx中：

```
import { ChangeEvent, useEffect,useState } from 'react'
    ...
    ...
    // 用户名，密码，验证码的处理
    const [usernameVal, setUsernameVal] = useState("");
    const [passwordVal, setPasswordVal] = useState("");
    const [captchaVal, setCaptchaVal] = useState("");
    const usernameChange = (e:ChangeEvent<HTMLInputElement>)=>{
        setUsernameVal(e.target.value)
    }
    const passwordChange = (e:ChangeEvent<HTMLInputElement>)=>{
        setPasswordVal(e.target.value)
    }
    const captchaChange = (e:ChangeEvent<HTMLInputElement>)=>{
        setCaptchaVal(e.target.value)
    }

    // 点击登录按钮的事件
    const gotoLogin = ()=>{
        console.log("用户输入的用户名和密码 还有验证码分别为：",
                    usernameVal,passwordVal,captchaVal);
    }
    ...
    ...

   <Input  placeholder="用户名" className='ipt' onChange={usernameChange}/>
   <Input.Password  placeholder="密码" onChange={passwordChange}
className='ipt'/>
   <div className='captchaBox'>
        <Input  placeholder="验证码" className='ipt' onChange={captchaChange}/>
        <div className="captchaImg">
            <img src="" height="38" alt="" />
        </div>
    </div>
    <Button className='loginBtn'  type='primary' block onClick={gotoLogin}>登录
</Button>
```

七、ReactRedux的基本配置

安装Redux和ReactRedux

```
npm i redux react-redux --save
```

/src下新建store，新建index.ts文件

```
import { legacy_createStore} from "redux";
import reducer from "./reducer";

// window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()
让浏览器redux-dev-tools能正常使用
const store = legacy_createStore(reducer,window.__REDUX_DEVTOOLS_EXTENSION__ &&
window.__REDUX_DEVTOOLS_EXTENSION__());

export default store
```

/src/store下新建reducer.ts文件

```
const defaultState = {
    num:20
}

let reducer = (state = defaultState, action:{type:string,val:number}) => {
    let newState = JSON.parse(JSON.stringify(state))

    switch (action.type) {
        case "add1":
            newState.num++;
            break;
        case "add2":
            newState.num +=action.val;
            break;
        default:
            break;
    }
    return newState
}
export default reducer
```

main.tsx中：

```
// 状态管理
import {Provider} from "react-redux";
import store from "./store";

ReactDOM.createRoot(document.getElementById('root')!).render(
  // <React.StrictMode>
  <Provider store={store}>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </Provider>
```

```
  // </React.StrictMode>
)
```

在Page1.tsx中：

```
import {useSelector,useDispatch} from "react-redux"


const View = () => {
  // 获取数据
  const { num } = useSelector((state) => ({
    num:state.num   // 这里划曲线警告
  }));
  // 修改数据
  const dispatch = useDispatch();
  const changeNum = () =>{
    dispatch({ type: 'add2',val:3 })
  }

  return(
    <div className='home'>
        <p>这是Page1页面内容</p>
        <p>{num}</p>
        <button onClick={changeNum}>按钮</button>
    </div>
  )
}

export default View
```

八、【重点】解决两个划线警告问题

8.1、解决获取redux数据num划曲线警告的问题

在Page1.tsx中：

```
import store from "@/store";
// TS中提供了 ReturnType ，用于 获取函数类型的返回值类型
type RootState = ReturnType<typeof store.getState>

...
    const { num } = useSelector((state:RootState) => ({
        num:state.num
    }));
```

但是每个模块都有可能用到这个类型，所以放到全局声明文件中去

新建types目录，新建store.d.ts文件：

```
// 【重点】d.ts 文件仅在没有任何导入时才被视为环境模块声明.如果您提供了一个导入行，它现在被视为一个普通的模块文件，而不是全局文件！！！
type RootState = ReturnType<typeof import('@/store').getState>;
```

【重点】d.ts 文件仅在没有任何导入时才被视为环境模块声明.

如果你提供了一个导入行，它就被视为一个普通的模块文件，而不是全局文件！！！

TypeScript在2.9以上版本中开始支持import()这种语法。

8.2、解决`Window.__REDUX_DEVTOOLS_EXTENSION__`划曲线报错的问题

在store.d.ts文件中继续加入：

```
interface Window{
    __REDUX_DEVTOOLS_EXTENSION__:function;
}
```

九、ReactRedux的数据和方法的抽离

9.1、初步抽离

按照功能模块文件夹来划分文件夹。

在/src/store下新建NumStatus文件夹，里面新建index.ts

```
export default {
    state: {
        num:20
    },
    actions: {
        add1(newState,action){
            newState.num++;
        },
        add2(newState,action){
            newState.num +=action.val;
        }
    },
}
```

在/src/store/reducer.ts中：

```
import handleNum from "./NumStatus";
const defaultState = {
    ...handleNum.state
    // 其他模块的数据
}

let reducer = (state = defaultState, action:{type:string,val:number}) => {
    let newState = JSON.parse(JSON.stringify(state))

    switch (action.type) {
        case "add1":
            handleNum.actions.add1(newState,action)
            break;
        case "add2":
            handleNum.actions.add2(newState,action)
            break;
        default:
            break;
```

```
    }
    return newState
}
export default reducer
```

9.2、方法名的统一管理

在/src/store/NumStatus/index.ts中：

```
export default {
    ...
    ...
    // 名字的统一管理
    add1:"add1",
    add2:"add2",
}
```

在/src/store/reducer.ts中：

```
switch (action.type) {
    case handleNum.add1:
        handleNum.actions[handleNum.add1](newState,action)  // 可能划红色曲线警告
        break;
    case handleNum.add2:
        handleNum.actions[handleNum.add2](newState,action)  // 可能划红色曲线警告
        break;
    // 其他模块的方法
    default:
        break;
}
```

解决handleNum.actions[handleNum.actionName2]的红色曲线警告：

我们在项目中的tsconfig.json文件中添加：`"suppressImplicitAnyIndexErrors":true,`选项，重启Vscode！

详细查看官方文档：https://www.typescriptlang.org/tsconfig#suppressImplicitAnyIndexErrors

```
"compilerOptions": {
    "suppressImplicitAnyIndexErrors":true,
    ...
}
```

十、【重点】ReactRedux的模块化配置

上面的模块化还不够彻底，如果模块多了reducer.js中就会有变得很长而且很乱，所有的模块都要在reducer.js中过一遍。

如何优把reducer也模块化到各个功能模块中？？ **利用combineReducers**

/src/store/index.ts

```
import { legacy_createStore,combineReducers } from "redux";

import handleArr from './ArrStatus/reducer';
```

```
import handleNum from './NumStatus/reducer';

// 组合各个模块的reducers
const reducers = combineReducers({
    handleArr,
    handleNum
});

// 创建仓库对象，注册reducers
const store = legacy_createStore(reducers, window.__REDUX_DEVTOOLS_EXTENSION__
&& window.__REDUX_DEVTOOLS_EXTENSION__());

export default store
```

在/src/store/NumStatus中新建reducer.ts

```
import handleNum from "./";


let reducer = (state = {...handleNum.state}, action:{type:string,val:number}) =>
{
    let newState = JSON.parse(JSON.stringify(state))

    switch (action.type) {
        case handleNum.add1:
            handleNum.actions[handleNum.add1](newState,action)
            break;
        case handleNum.add2:
            handleNum.actions[handleNum.add2](newState,action)
            break;
        default:
            break;
    }
    return newState
}
export default reducer
```

在配置多一个模块：

新建/src/store/中ArrStatus文件夹，下新建index.ts和reducer.ts

/src/store/ArrStatus/index.ts中：

```
export default {
    state: {
        sarr:[10,20,30]
    },
    actions: {
        sarrpush(newState,action){
            newState.sarr.push(action.val)
        },

    },
    sarrpush:"sarrpush",
}
```

/src/store/ArrStatus/reducer.ts中：

```
import handleArr from "./"

let reducer = (state = {...handleArr.state}, action:{type:string,val:number}) =>
{
    let newState = JSON.parse(JSON.stringify(state))

    switch (action.type) {
        case handleArr.sarrpush:
            handleArr.actions[handleArr.sarrpush](newState,action)
            break;
        default:
            break;
    }
    return newState
}
export default reducer
```

最后需要在组件中修改获取数据的方式，才能在界面上正常看到数据：

```
// 获取数据
const { num } = useSelector((state:RootState) => ({
    num:state.handleNum.num
}));

...
 // 获取sarr数据
  const { sarr } = useSelector((state:RootState) => ({
    sarr:state.handleArr.sarr
  }));
  // 修改sarr数据
  const changeArr = () =>{
    dispatch({ type: 'sarrpush',val:3 })
  }

  return(
    <div className='home'>
        <p>这是Page1页面内容</p>
        <p>{num}</p>
        <button onClick={changeNum}>按钮</button>
        <hr />
        <p>{sarr}</p>
        <button onClick={changeArr}>按钮2</button>
    </div>
  )
```

十一、【重点】ReactRedux代码优化

11.1、switch...case...语句自动生成

对于reduce.ts每次写一个方法，就要手动添加这个case的对比和执行，相当麻烦！

/src/store/NumStatus/index.ts中：

```
export default {
    state: {
```

```
        num:20
    },
    actions: {
        add1(newState:{num:number},action:{type:string}){
            newState.num++;
        },
        add2(newState:{num:number},action:{type:string,val:number}){
            newState.num +=action.val;
        }
    },
    // 名字的统一管理
    // add1:"add1",
    // add2:"add2",
    actionNames:{
        add1:"add1",
        add2:"add2",
    }
}
```

/src/store/NumStatus/reducer.ts中:

```
// switch (action.type) {
//     case handleNum.actionName1:
//         handleNum.actions[handleNum.add1](newState,action)
//         break;
//     case handleNum.actionName2:
//         handleNum.actions[handleNum.add2](newState,action)
//         break;
//     default:
//         break;
// }
// 【优化】这样就每次写一个方法就不需要手动来添加这句case和执行，终于解放双手了！！
for(let key in handleNum.actionNames){
    if(action.type===handleNum.actionNames[key]){
        handleNum.actions[handleNum.actionNames[key]](newState,action)
        break
    }
}
```

11.2、方法名对象actionNames自动生成

/src/store/NumStatus/index.ts中:

```
const store = {
    state: {
        num:21
    },
    actions: {
        add1(newState:{num:number},action:{type:string}){
            newState.num++;
        },
        add2(newState:{num:number},action:{type:string,val:number}){
            newState.num +=action.val;
        }
    },
    // 名字的统一管理
```

```
    // add1:"add1",
    // add2:"add2",
    // actionNames:{
    //     add1:"add1",
    //     add2:"add2",
    // }
    actionNames:{}
}
// 自动生成 actionNames{add1:"add1", add2:"add2"} 对象
let actionNames = {}
for(let key in store.actions){
    actionNames[key]=key
}
store.actionNames=actionNames
export default store
```

11.3、完善各个模块的reducer和ArrStatus的代码

所有模块的reducer最终处理成：

```
import handler from "./";

let reducer = (state = {...handler.state}, action:{type:string}) => {
    let newState = JSON.parse(JSON.stringify(state))
    for(let key in handler.actionNames){
        if(action.type===handler.actionNames[key]){
            handler.actions[handler.actionNames[key]](newState,action)
            break
        }
    }
    return newState
}
export default reducer
```

src/store/ArrStatus/index.ts最终也处理成：

```
const store = {
    state: {
        sarr:[10,20,30]
    },
    actions: {
        sarrpush(newState:{sarr:number[]},action:{type:string,val:number}){
            newState.sarr.push(action.val)
        },
    },
    actionNames:{}
}
let actionNames = {}
for(let key in store.actions){
    actionNames[key]=key
}
store.actionNames=actionNames
export default store
```