环境

```
node:
16.17.1

npm:
8.15.0
```

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

一、react-redux的异步解决方案redux-thunk

在store/NumStatus/index.ts中做异步操作：

```
    add1(newState:{num:number},action:{type:string}){
        //newState.num++;
        // 会有bug 没有办法达到延迟和修改的效果
        setTimeout(()=>{
            newState.num++;
        },1000)

    },
```

会发现这种写法其实达不到想要的异步效果，需要通过redux相关的异步方案来解决（市面上有redux-saga，redux-thunk），今天我们使用redux-thunk来做。

redux-thunk相比于redux-saga，体积小，灵活，但需要自己手动抽取和封装。但学习成本较低。

项目目录下安装redux-thunk

```
npm i redux-thunk
```

在store/index.ts中：

```
import { legacy_createStore,combineReducers,applyMiddleware,compose } from
"redux"; //rt
import reduxThunk from 'redux-thunk' //rt

import handleArr from './ArrStatus/reducer.ts';
import handleNum from './NumStatus/reducer.ts';

// 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__());

// 判断有没有__REDUX_DEVTOOLS_EXTENSION_COMPOSE__这个模块
```

```
let composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ ?
window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__({}):compose //rt

// 把仓库数据，浏览器redux-dev-tools，还有reduxThunk插件关联在store中
const store =
legacy_createStore(reducers,composeEnhancers(applyMiddleware(reduxThunk))); //rt
export default store
export default store
```

在src/views/Page1.tsx组件中：

```
const changeNum2 = () =>{
    // redux-thunk的写法：
    dispatch((dis:Function)=>{
        setTimeout(()=>{
            dis({type:'redu1',val:n})
        },1000);
    });
}

// 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}>同步按钮</button>
      <button onClick={changeNum2}>异步按钮</button>
```

但上面这样写就把数据操作和项目业务混在一起了，这也是redux-think本身的缘故，需要自己手动优化封装函数进行优化！！！

二、手动封装redux-thunk的异步函数

上面redux-thunk的写法可以看出，dispatch调用传入的是一个actions函数，接下来我们把这个函数抽取到状态管理中作为一个方法的返回值。

在store/Numstatus/index.ts中的store对象中添加：

```
  // 优化redux-thunk的异步写法(模仿Vuex的写法)
  asyncActions:{  // 只放异步的方法
      asyncAdd1(dispatch:Function){
          setTimeout(()=>{
              dispatch({type:"add1"})
          },1000)
      }
  },
```

而src/views/Page1.tsx组件中的changeNum2修改成：

```
import numStatus from "@/store/NumStatus"
...
...
const changeNum2 = () =>{

    // redux-thunk的写法：
    // dispatch((dis:Function)=>{
    //     setTimeout(()=>{
    //         dis({type:'redu1',val:n})
    //     },1000);
    // });
    // 但上面这样写就把数据操作和项目业务混在一起了
    // 优化redux-thunk的异步写法  `
    // dispatch(调用状态管理中的asyncAdd1)
    dispatch(numStatus.asyncActions.asyncAdd1)
}
```

至此，react状态管理模块的封装完成！

三、数据交互的解决方案

3.1、axios封装和apis的抽取

安装axios

```
npm i axios
```

/src下新建request文件夹，并新建index.ts

```
import axios from "axios"

// 创建axios实例
const instance = axios.create({
    // 基本请求路径的抽取
    baseURL:"http://xue.cnkdl.cn:23683",
    // 这个时间是你每次请求的过期时间，这次请求认为20秒之后这个请求就是失败的
    timeout:20000
})

// 请求拦截器
instance.interceptors.request.use(config=>{

    return config
},err=>{
    return Promise.reject(err)
});

// 响应拦截器
instance.interceptors.response.use(res=>{

    return res.data
},err=>{
    return Promise.reject(err)
})
```

```
export default instance
```

/src/request下新建api.ts

```
// 统一管理项目中所有的请求路径 api
import request from "./index"

// 验证码请求
export const captchaAPI = () => request.get("/prod-api/captchaImage");
```

Login.tsx中：

```
import { captchaAPI } from "@/request/api.ts"
...
...
    // 点击验证码图片的事件函数
    const getCaptchaImg = ()=>{
        captchaAPI().then(res=>{    // 划红色曲线警告
            console.log(res);
        })
    }
```

四、res划红色曲线警告的解决

res必须为请求回来的类型的数据，需要添加类型。

建议安装 VSCode的 JSON to TS 扩展。 (快捷键：ctrl+shift+alt+v)

在浏览器中复制请求到的数据，粘贴，按下ctrl+shift+alt+v，就能得到类型该res数据的类型。起个名字就可以用了！

在src/types文件夹中新建api.d.ts，把上面生成的类型粘贴进去：

```
// 验证码
interface CaptchAPIRes {
    msg: string;
    img: string;
    code: number;
    captchaEnabled: boolean;
    uuid: string;
}
```

Login.tsx中：

```
const getCaptchaImg = ()=>{
    captchaAPI().then((res:CaptchAPIRes)=>{
        console.log(res);
    })
}
```

把上面代码换成async+await的写法：

```
const getCaptchaImg = async ()=>{
    // CaptchaAPI().then((res:CaptchAPIRes)=>{
    //     console.log(res.code);
    // })
    let captchAPIRes = await CaptchaAPI()
    console.log(captchAPIRes.code);

}
```

五、规范化请求中TypeScript的书写

请求中请求参数和函数类型都需要进行约束!

src/request/api.ts中:

```
// 统一管理项目中所有的请求路径 api
import request from "./index"

// 验证码请求
export const CaptchaAPI = ():Promise<CaptchAPIRes> => request.get("/prod-
api/captchaImage");
```

六、完成点击验证码业务

Login组件中:

```
const [captchaImg,setCaptchaImg] = useState("");

  useEffect(()=>{
    // 加载完这个组件之后，加载背景
    initLoginBg();
    window.onresize = function(){initLoginBg()};

    getCaptchaImg()
  },[]);


// 点击验证码图片的事件函数
const getCaptchaImg = async ()=>{

    let captchAPIRes:CaptchAPIRes = await CaptchaAPI()

    // 1、把图片显示在img上面
    setCaptchaImg(""+captchAPIRes.img)
    // 2、获取到验证码之后需要保存uuid
    localStorage.setItem("uuid",captchAPIRes.uuid)

}
```

七、测试登录请求

/src/request/api.ts中:

```
export const loginAPI = (params:LoginAPIReq) => request.post("/prod-
api/login",params);
```

/src/types/api.d.ts中：

```
// 登录请求类型
interface LoginAPIReq {
  username: string;
  password: string;
  code: string;
  uuid: string;
}
// 登录响应类型
interface LoginAPIRes {
  msg: string;
  code: number;
  token: string;
}
```

Login.tsx中：

```
import { Input,Space,Button,message} from 'antd';
import { CaptchaAPI,loginAPI } from "@/request/api.ts"
import { useNavigate } from "react-router-dom"

....
const [captchaImg,setCaptchaImg] = useState("");
const navigateTo = useNavigate()
....
// 点击登录按钮的事件函数
const gotoLogin = async ()=>{
    // console.log("用户输入的用户名，密码，验证码分别
是：",usernameVal,passwordVal,captchaVal);
    if(!usernameVal.trim()|| !passwordVal.trim()|| !captchaVal.trim()){
      message.warning('请完整输入信息！');
      return
    }

    // 发起登录请求
    let loginAPIRes:LoginAPIRes = await loginAPI({
      username: usernameVal,
      password: passwordVal,
      code: captchaVal,
      uuid: localStorage.getItem("uuid")
    })
    console.log(loginAPIRes);

    if(loginAPIRes.code===200){
      // 1、提示操作成功
      message.success('登录成功！');
      // 2、保存token
      localStorage.setItem("lege-react-management-token",loginAPIRes.token)
      // 3、跳转到/page1页面
      navigateTo("/page1")
    }
```

```
}
```

八、手动封装前置路由守卫

App.tsx中：

```
import { useEffect } from 'react'
import { useRoutes, useLocation,useNavigate } from "react-router-dom"
import router from "./router"
import { message } from "antd"

// 去往登录页的组件
function ToLogin(){
  const navigateTo = useNavigate()
  // 加载完这个组件之后实现跳转
  useEffect(()=>{
    // 加载完组件之后执行这里的代码
    navigateTo("/login");
    message.warning("您还没有登录，请登录后再访问！");
  },[])
  return <div></div>
}
// 去往首页的组件
function ToPage1(){
  const navigateTo = useNavigate()
  // 加载完这个组件之后实现跳转
  useEffect(()=>{
    // 加载完组件之后执行这里的代码
    navigateTo("/page1");
    message.warning("您已经登录过了！");
  },[])
  return <div></div>
}
// 手写封装路由守卫
function BeforeRouterEnter(){
  const outlet = useRoutes(router);

  /*
    后台管理系统两种经典的跳转情况：
    1、如果访问的是登录页面， 并且有token， 跳转到首页
    2、如果访问的不是登录页面，并且没有token， 跳转到登录页
    3、其余的都可以正常放行
  */
    const location = useLocation()
    let token = localStorage.getItem("lege-react-management-token");
    //1、如果访问的是登录页面， 并且有token， 跳转到首页
    if(location.pathname==="/login" && token){
      // 这里不能直接用 useNavigate 来实现跳转 ，因为需要BeforeRouterEnter是一个正常的
JSX组件
      return <ToPage1 />
    }
    //2、如果访问的不是登录页面，并且没有token， 跳转到登录页
    if(location.pathname!=="/login" && !token){
      return <ToLogin />
    }
```

```
    return outlet
}


function App() {
  return (
    <div className="App">

      {/* <Link to="/home">Home</Link> |
      <Link to="/about">About</Link> |
      <Link to="/user">User</Link> */}

      {/* 占位符组件，类似于窗口，用来展示组件的，有点像vue中的router-view */}
      {/* <Outlet></Outlet> */}
      {/* {outlet} */}
      <BeforeRouterEnter />
    </div>
  )
}

export default App
```