---

link: null
title: 珠峰架构师成长计划
description: public\index.html
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=179 sentences=372, words=3564

1.Redux Toolkit 是什么？

- Redux Toolkit (https://redux-toolkit.js.org)是我们官方的，有观点的，开箱即用的高效 Redux 开发工具集
- Redux Toolkit解决的问题
    - 配置 Redux store 过于复杂
    - 我必须添加很多软件包才能开始使用 Redux 做事情
    - Redux 有太多样板代码

2. 安装

```
npm install redux redux-logger redux-thunk @reduxjs/toolkit   express cors  axios --save
```

3. 正常用法

public\index.html

```
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <title>Redux Toolkittitle>
  head>
  <body>
    <div>
      <p id="value">0p>
      <button id="add">+button>
      <button id="minus">-button>
    div>
  body>
html>
```

src\index.js

```
import { createStore } from 'redux';
const ADD = 'ADD'
const MINUS = 'MINUS'

function add() {
    return { type: ADD }
}

function minus() {
    return { type: MINUS }
}
function counter(state = { number: 0 }, action) {
    switch (action.type) {
        case ADD:
            return {number:state.number+1}
        case MINUS:
            return {number:state.number-1}
        default:
            return state
    }
}

var store = createStore(counter)
var valueEl = document.getElementById('value')

function render() {
    valueEl.innerHTML = store.getState().number;
}

render()
store.subscribe(render)

document.getElementById('add').addEventListener('click', function () {
    store.dispatch(add())
})

document.getElementById('minus').addEventListener('click', function () {
    store.dispatch(minus())
})
```

- action creator增加代码量
- const ADD = 'ADD'冗余
- switch结构不清晰

4. configureStore

- Redux工具包有一个 configureStore() 函数，其中覆盖了 createStore() 的功能
- configureStore() 函数 提供简化的配置选项。它可以自动组合切片 slice 的 reducer，添加你提供的任何 Redux 中间件，默认 情况下包含 redux-thunk，并启用Redux DevTools 扩展 (https://github.com/zalmoxisus/redux-devtools-extension)

```
    Redux Toolkit

      0
      +
      -
+     async-add
```

src\index.js

```
-import { createStore } from 'redux';
+import {configureStore} from '@reduxjs/toolkit';
+import {configureStore} from './toolkit';
+import thunk from 'redux-thunk';
+import logger from 'redux-logger';
const ADD = 'ADD'
const MINUS = 'MINUS'

function add() {
    return { type: ADD }
}

function minus() {
    return { type: MINUS }
}
function counter(state = { number: 0 }, action) {
    switch (action.type) {
        case ADD:
            return {number:state.number+1}
        case MINUS:
            return {number:state.number-1}
        default:
            return state
    }
}

-let store = createStore(counter)
+const store = configureStore({
+  reducer: counter,
+  middleware: [thunk, logger]
+})
var valueEl = document.getElementById('value')

function render() {
    valueEl.innerHTML = store.getState().number;
}

render()
store.subscribe(render)

document.getElementById('add').addEventListener('click', function () {
    store.dispatch(add())
})

document.getElementById('minus').addEventListener('click', function () {
    store.dispatch(minus())
})
+document.getElementById('async-add').addEventListener('click', function () {
+  store.dispatch((dispatch)=>{
+    setTimeout(()=>{
+      dispatch(add())
+    },1000)
+  })
+})
```

src\toolkit\index.js

```
export { default as configureStore } from './configureStore';
```

src\toolkit\configureStore.js

```
import { combineReducers,applyMiddleware,createStore,compose} from 'redux';
function isPlainObject(value) {
    if (typeof value !== "object" || value === null)
        return false;
    return Object.getPrototypeOf(value) === Object.prototype;
}
function configureStore(options = {}) {
    let { reducer, middleware, preloadedState } = options;
    let rootReducer;
    if (typeof reducer === "function") {
        rootReducer = reducer;
    } else if (isPlainObject(reducer)) {
        rootReducer = combineReducers(reducer);
    }
    const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
    return createStore(rootReducer, preloadedState, composeEnhancers(enhancer));
}
export default configureStore;
```

compose

```
function add1(str){
    return str+1;
}
function add2(str){
    return str+2;
}
function compose(fn1,fn2){
 return function(str){
   return fn1(fn2(str))
 }
}
let fn = compose(add1, add2);
let result = fn('zhufeng');
console.log(result);
```

5. createAction

- createAction 接受一个 action 类型字符串作为参数，并返回一个使用该类型字符串的 action creator 函数

src\index.js

```
+import {configureStore,createAction} from './toolkit';
import thunk from 'redux-thunk';
import logger from 'redux-logger';
-const ADD = 'ADD'
-const MINUS = 'MINUS'

-function add() {
-    return { type: ADD }
-}
+const add = createAction('ADD')
-function minus() {
-    return { type: MINUS }
-}
+const minus = createAction('MINUS', (amount) => ({ payload: amount }))
+console.log(minus.toString());
+console.log(minus.type);

function counter(state = { number: 0 }, action) {
    switch (action.type) {
+       case add.type:
          return {number:state.number+1}
+       case minus.type:
+         return {number:state.number-action.payload}
        default:
            return state
    }
}

const store = configureStore({
  reducer: counter,
  middleware: [thunk, logger]
})
var valueEl = document.getElementById('value')

function render() {
    valueEl.innerHTML = store.getState().number;
}

render()
store.subscribe(render)

document.getElementById('add').addEventListener('click', function () {
    store.dispatch(add())
})

document.getElementById('minus').addEventListener('click', function () {
+   store.dispatch(minus(2))
})
document.getElementById('async-add').addEventListener('click', function () {
  store.dispatch((dispatch)=>{
    setTimeout(()=>{
      dispatch(add())
    },1000)
  })
})
```

src\toolkit\index.js

```
export { default as configureStore } from './configureStore';
+export { default as createAction } from './createAction';
```

src\toolkit\createAction.js

```
function createAction(type, prepareAction) {
    function actionCreator(...args) {
        if (prepareAction) {
            var prepared = prepareAction.apply(null, args);
            return {
                type: type,
                payload: prepared.payload,
                error: prepared.error
            };
        }
        return {
            type: type,
            payload: args[0]
        };
    }
    actionCreator.toString = function () {
        return "" + type;
    }
    actionCreator.type = type;
    return actionCreator;
}
export default createAction;
```

6. createReducer

- Redux工具包 包含了一个 createReducer 函数，它让使用"查找表"对象的方式编写 reducer
- 其中对象的每一个 key 都是一个 Redux action type 字符串，value 是 reducer 函数

src\index.js

```
import {configureStore,createAction,createReducer} from './toolkit';
import thunk from 'redux-thunk';
import logger from 'redux-logger';
const add = createAction('ADD')
const minus = createAction('MINUS', (amount) => ({ payload: amount }))
console.log(minus.toString());
console.log(minus.type);

-function counter(state = { number: 0 }, action) {
-    switch (action.type) {
-        case add.type:
-            return {number:state.number+1}
-        case minus.type:
-            return {number:state.number-action.payload}
-        default:
-            return state
-    }
-}

+const counter = createReducer({number:0}, {
+  [add]: state => ({number:state.number+1}),
+  [minus]: state => ({number:state.number-1})
+})

const store = configureStore({
  reducer: counter,
  middleware: [thunk, logger]
})
var valueEl = document.getElementById('value')

function render() {
    valueEl.innerHTML = store.getState().number;
}

render()
store.subscribe(render)

document.getElementById('add').addEventListener('click', function () {
    store.dispatch(add())
})

document.getElementById('minus').addEventListener('click', function () {
    store.dispatch(minus(2))
})
document.getElementById('async-add').addEventListener('click', function () {
  store.dispatch((dispatch)=>{
    setTimeout(()=>{
      dispatch(add())
    },1000)
  })
})
```

src\toolkit\createReducer.js

```
function createReducer(initialState, reducers={}) {
    return function (state = initialState, action) {
        let reducer = reducers[action.type];
        if (reducer) return reducer(state, action);
        return state;
    }
}
export default createReducer;
```

src\toolkit\index.js

```
export { default as configureStore } from './configureStore';
export { default as createAction } from './createAction';
+export { default as createReducer } from './createReducer';
```

7. createSlice

- createSlice函数允许我们提供一个带有reducer函数的对象，并且它将根据我们列出的 reducer 的名称自动生成 action type 字符串和 action creator 函数
- createSlice 返回一个 分片对象，该对象包含生成的 reducer 函数作为一个名为 reducer 的字段，以及在一个名为 actions 的对象中生成的 action creator

    - reducers 一个包含case reducer函数的对象，它的key将被用来生成动作类型常量并在派发的时候可见
    - prepare 可以用来自定义payload的值的创建
    - extraReducers 允许createSlice去响应别的slice创建的动作类型，它们不会用来生成actions

```
//import {configureStore,createAction,createReducer,createSlice} from '@reduxjs/toolkit';
import {configureStore,createAction,createReducer,createSlice} from './toolkit';
import thunk from 'redux-thunk';
import logger from 'redux-logger';
-const add = createAction('ADD')
-const minus = createAction('MINUS', (amount) => ({ payload: amount }))
-const counter = createReducer({number:0}, {
-  [add]: state => ({number:state.number+1}),
-  [minus]: state => ({number:state.number-1})
-})

+const counterSlice = createSlice({
+  name: 'counter',
+  initialState: {number:0},
+  reducers: {
+    add: (state)  =>  ({number:state.number+1}),//派发的时候动作类型是 counter/add
+    minus: (state,action) => ({number:state.number-action.payload})
+  }
+})
const { actions, reducer } = counterSlice
console.log(actions);
const { add, minus } = actions
console.log(add);

const store = configureStore({
  reducer: reducer,
  middleware: [thunk, logger]
})
var valueEl = document.getElementById('value')

function render() {
    valueEl.innerHTML = store.getState().number;
}

render()
store.subscribe(render)

document.getElementById('add').addEventListener('click', function () {
    store.dispatch(add())
})

document.getElementById('minus').addEventListener('click', function () {
    store.dispatch(minus(2))
})
document.getElementById('async-add').addEventListener('click', function () {
  store.dispatch((dispatch)=>{
    setTimeout(()=>{
      dispatch(add())
    },1000)
  })
})
```

src\toolkit\createSlice.js

```
import { createReducer, createAction } from './'
function createSlice(options) {
    let { name, initialState={}, reducers={} } = options;
    let actions = {};
    const prefixReducers = {};
    Object.keys(reducers).forEach(function (key) {
        var type = getType(name, key);
        actions[key] = createAction(type);
        prefixReducers[type]=reducers[key];
    })
    let reducer = createReducer(initialState, prefixReducers);
    return {
        name,
        reducer,
        actions
    };
}
function getType(slice, actionKey) {
    return slice + "/" + actionKey;
}
export default createSlice;
```

src\toolkit\index.js

```
export { default as configureStore } from './configureStore';
export { default as createAction } from './createAction';
export { default as createReducer } from './createReducer';
+export { default as createSlice } from './createSlice';
```

8. immer

- 对 draftState 的修改都会反应到 nextState 上
- 而 immer 使用的结构是共享的，nextState 在结构上又与 currentState 共享未修改的部分
    - currentState 被操作对象的最初状态
    - draftState 根据 currentState 生成的草稿状态，它是 currentState 的代理，对 draftState 所做的任何修改都将被记录并用于生成 nextState 。在此过程中，currentState 将不受影响
    - nextState 根据 draftState 生成的最终状态
    - produce 生产 用来生成 nextState的函数

```
let produce = require('immer').default;
let baseState = {
    ids: [1],
    pos: {
      x: 1,
      y: 1
    }
  }

  let nextState = produce(baseState, (draft) => {
    draft.ids.push(2);
  })
  console.log(baseState.ids === nextState.ids);
  console.log(baseState.pos === nextState.pos);
```

src\index.js

```
//import {configureStore,createAction,createReducer,createSlice} from '@reduxjs/toolkit';
import {configureStore,createAction,createReducer,createSlice} from './toolkit';
import thunk from 'redux-thunk';
import logger from 'redux-logger';
const counterSlice = createSlice({
  name: 'counter',
  initialState: {number:0},
  reducers: {
+   add: (state)  =>  state.number+=1,//派发的时候动作类型是 counter/add
+   minus: (state,action) => state.number-=action.payload
  }
})
const { actions, reducer } = counterSlice
console.log(actions);
const { add, minus } = actions
console.log(add);

const store = configureStore({
  reducer: reducer,
  middleware: [thunk, logger]
})
var valueEl = document.getElementById('value')

function render() {
    valueEl.innerHTML = store.getState().number;
}

render()
store.subscribe(render)

document.getElementById('add').addEventListener('click', function () {
    store.dispatch(add())
})

document.getElementById('minus').addEventListener('click', function () {
    store.dispatch(minus(2))
})
document.getElementById('async-add').addEventListener('click', function () {
  store.dispatch((dispatch)=>{
    setTimeout(()=>{
      dispatch(add())
    },1000)
  })
})
```

src\toolkit\createReducer.js

```
+import produce from 'immer';
function createReducer(initialState, reducers={}) {
    return function (state = initialState, action) {
        let reducer = reducers[action.type];
+       if (reducer)
+           return produce(state, draft => {
+               reducer(draft, action);
+           });
        return state;
    }
}
export default createReducer;
```

9. reselect

- reselect可以缓存运算结果，提升性能
- reselect的原理是，只要相关状态不变，即直接使用上一次的缓存结果

```
function createSelector(selectors, reducer) {
    let lastState;
    let lastValue;
    return function (state) {
        if (lastState === state) {
            return lastValue;
        }
        let values = selectors.map(selector => selector(state));
        lastValue = reducer(...values);
        lastState = state;
        return lastValue;
    }
}
const selectCounter1 = state => state.counter1
const selectCounter2 = state => state.counter2
const totalSelector = createSelector(
    [selectCounter1, selectCounter2],
    (counter1, counter2) => {
        console.log('计算结果');
        return counter1.number + counter2.number;
    }
)
let state = { counter1: { number: 1 }, counter2: { number: 2 } };
let state1 = totalSelector(state);
console.log(state1);
let state2 = totalSelector(state);
console.log(state2);
```

public\index.html

```
    Redux Toolkit

+     0
+     add1
+     minus1
+
+     0
+     add2
+     minus2
+
+     0
```

src\index.js

```
//import {configureStore,createAction,createReducer,createSlice} from '@reduxjs/toolkit';
import {configureStore,createAction,createReducer,createSlice,createSelector} from './toolkit';
import thunk from 'redux-thunk';
import logger from 'redux-logger';

const counter1Slice = createSlice({
  name: 'counter1',
  initialState: { number: 0 },
  reducers: {
    add: state => { state.number += 1 },
    minus: state => { state.number -= 1 }
  }
})
const counter2Slice = createSlice({
  name: 'counter2',
  initialState: { number: 0 },
  reducers: {
    add: state => { state.number += 1 },
    minus: state => { state.number -= 1 }
  }
})
const { actions: { add: add1, minus: minus1 }, reducer: reducer1 } = counter1Slice
const { actions: { add: add2, minus: minus2 }, reducer: reducer2 } = counter2Slice

const store = configureStore({
  reducer: { counter1: reducer1, counter2: reducer2 },
  middleware: [thunk, logger]
})
var value1El = document.getElementById('value1')
var value2El = document.getElementById('value2')
var sumEl = document.getElementById('sum')
const selectCounter1 = state => state.counter1
const selectCounter2 = state => state.counter2
const totalSelector = createSelector(
  [selectCounter1, selectCounter2],
  (counter1, counter2) => {
    return counter1.number + counter2.number;
  }
)
function render() {
  value1El.innerHTML = store.getState().counter1.number;
  value2El.innerHTML = store.getState().counter2.number;
  sumEl.innerHTML = totalSelector(store.getState());
}

render()
store.subscribe(render)

document.getElementById('add1').addEventListener('click', function () {
  store.dispatch(add1())
})

document.getElementById('minus1').addEventListener('click', function () {
  store.dispatch(minus1())
})

document.getElementById('add2').addEventListener('click', function () {
  store.dispatch(add2())
})

document.getElementById('minus2').addEventListener('click', function () {
  store.dispatch(minus2())
})
```

src\toolkit\index.js

```
export { default as configureStore } from './configureStore';
export { default as createAction } from './createAction';
export { default as createReducer } from './createReducer';
export { default as createSlice } from './createSlice';
+export {createSelector } from '../reselect';
```

src\reselect\index.js

```
export {default as createSelector} from './createSelector';
```

src\reselect\createSelector.js

```
function createSelector(selectors, reducer) {
    let lastState;
    let lastValue;
    return function (state) {
        if (lastState === state) {
            return lastValue;
        }
        let values = selectors.map(selector => selector(state));
        lastValue = reducer(...values);
        lastState = state;
        return lastValue;
    }
}
export default createSelector;
```

10. createAsyncThunk

- 接收redux动作类型字符串和一个返回promise回调的函数
- 它会基于你传递的动作类型前缀生成promise生命周期的动作类型
- 并且返回一个thunk动作创建者，这个thunk动作创建者会运行promise回调并且派发生命周期动作
- 它抽象了处理异步请求生命周期的标准推荐方法

src\index.js

```
import { configureStore, createSlice, createAsyncThunk } from './toolkit';
import axios from 'axios';

export const getTodosList = createAsyncThunk(
  "todos/list",  async () =>  await axios.get(`http://localhost:8080/todos/list`)
);

const initialState = {
  todos: [],
  loading: false,
  error: null,
};

const todoSlice = createSlice({
  name: 'todo',
  initialState,
  reducers: {},
  extraReducers: {
    [getTodosList.pending]: (state) => {
      state.loading = true;
    },
    [getTodosList.fulfilled]: (state, action) => {
      state.todos = action.payload.data;
      state.loading = false;
    },
    [getTodosList.rejected]: (state, action) => {
      state.todos = [];
      state.error = action.error.message;
      state.loading = false;
    }
  }
})
const { reducer } = todoSlice;
const store = configureStore({
  reducer
})
let promise = store.dispatch(getTodosList());
console.log('请求开始',store.getState());

promise.then((response)=>{
  console.log('成功',response);
  setTimeout(()=>{
    console.log('请求结束',store.getState());
  },);
},error=>{
  console.log('失败',error);
  setTimeout(()=>{
    console.log('请求结束',store.getState());
  },);
});
```

src\toolkit\configureStore.js

```
import { combineReducers,applyMiddleware,createStore,compose } from 'redux';
+import thunk from 'redux-thunk';
function isPlainObject(value) {
    if (typeof value !== "object" || value
        return false;
    return Object.getPrototypeOf(value)
}
function configureStore(options = {}) {
+   let { reducer, middleware=[thunk], preloadedState } = options;
    let rootReducer;
    if (typeof reducer
        rootReducer = reducer;
    } else if (isPlainObject(reducer)) {
        rootReducer = combineReducers(reducer);
    }
    const enhancer = applyMiddleware(...middleware);
    const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
    return createStore(rootReducer, preloadedState, composeEnhancers(enhancer));
}
export default configureStore;
```

src\toolkit\createReducer.js

```
import produce from 'immer';
+function createReducer(initialState, reducers={}, extraReducers={}) {
    return function (state = initialState, action) {
        let reducer = reducers[action.type];
        if (reducer)
            return produce(state, draft => {
                reducer(draft, action);
            });
+       let extraReducer = extraReducers[action.type];
+       if (extraReducer) {
+           return produce(state, draft => {
+               extraReducer(draft, action);
+           });
+       }
        return state;
    }
}
export default createReducer;
```

src\toolkit\createSlice.js

```
import { createReducer, createAction } from './'
function createSlice(options) {
+   let { name, initialState={}, reducers={},extraReducers={}  } = options;
    let actions = {};
    const prefixReducers = {};
    Object.keys(reducers).forEach(function (key) {
        var type = getType(name, key);
        actions[key] = createAction(type);
        prefixReducers[type]=reducers[key];
    })
+   let reducer = createReducer(initialState, prefixReducers,extraReducers);
    return {
        name,
        reducer,
        actions
    };
}
function getType(slice, actionKey) {
    return slice + "/" + actionKey;
}
export default createSlice;
```

src\toolkit\createAsyncThunk.js

```
import { createAction } from './';
function createAsyncThunk(typePrefix, payloadCreator) {
    let pending = createAction(typePrefix + "/pending", function () {
        return ({ payload: void 0 });
    });
    let fulfilled = createAction(typePrefix + "/fulfilled", function (payload) {
        return ({ payload });
    });
    let rejected = createAction(typePrefix + "/rejected", function (error) {
        return ({ error });
    });

    function actionCreator(arg) {
        return function (dispatch) {
            dispatch(pending());
            const promise = payloadCreator(arg);
            let abort;
            const abortedPromise = new Promise((_, reject) => {
                abort = () => {
                    reject({ name: "AbortError", message: "Aborted" });
                }
            });
            Promise.race([promise, abortedPromise]).then(result => {
                return dispatch(fulfilled(result));
            }, (error) => {
                return dispatch(rejected(error));
            });
            return Object.assign(promise, { abort });
        }
    }
    return Object.assign(actionCreator, { pending, rejected, fulfilled });
}
export default createAsyncThunk;
```

src\toolkit\index.js

```
export { default as configureStore } from './configureStore';
export { default as createAction } from './createAction';
export { default as createReducer } from './createReducer';
export { default as createSlice } from './createSlice';
export {createSelector } from '../reselect';
+export { default as createAsyncThunk } from "./createAsyncThunk";
```

api.js

```
let express = require('express');
let cors = require('cors');
let app = express();
app.use(cors());
app.use((req,res,next)=>{
    setTimeout(()=>{
      if(Math.random()>.5){
        next();
      }else{
        next('接口出错');
      }
    },1000);
});
let todos = [{id:1,text:"吃饭"},{id:2,text:"睡觉"}];
app.get('/todos/list',(_req,res)=>{
  res.json(todos);
});
app.get('/todos/detail/:id',(req,res)=>{
  let id = req.params.id;
  let todo = todos.find(item=>item.id === parseInt(id));
  res.json(todo);
});
app.listen(8080,()=>console.log(`服务在端口8080启动`));
```

11.Redux Toolkit Query

- Redux Toolkit Query (https://redux-toolkit.js.org/rtk-query/overview)是一种高级的数据获取和缓存工具，旨在简化在Web应用程序中加载数据的常见情况
- 跟踪加载状态用来显示UI转圈组件
- 避免对相同的数据进行重复请求
- 优化UI的更新感觉更快
- 根据用户的交互来管理缓存的生命周期
- createApi() RTK Query的核心函数，它允许你定义endpoint的集合用来描述如何获取数据，包含如何获取和转换数据
- fetchBaseQuery() 一个用来简化请求的对fetch的封装

src\index.js

```
    Redux Toolkit

+
```

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import store from './store';
import App from './App';
ReactDOM.render(<Provider store={store}><App />Provider>, document.getElementById('root'));
```

src\App.js

```
import todosApi from './todos'

function App() {

    const { data, error, isLoading } = todosApi.endpoints.getTodos.useQuery(1)

    console.log('isLoading=', isLoading, 'error=', error, 'data=', data);
    if(isLoading){
        return <div>加载中....div>;
    }else{
        if(error){
            return <div>{error.error}div>;
        }else if(data){
            return <div>{data.text}div>;
        }else{
            return null;
        }
    }
}
export default App;
```

src\store.js

```
import { configureStore } from '@reduxjs/toolkit'

import todosApi from './todos'

const store = configureStore({
  reducer: {
    [todosApi.reducerPath]: todosApi.reducer,
  },
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware().concat(todosApi.middleware)
})
export default store;
```

src\todos.js

```

import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react'

const todosApi = createApi({
  reducerPath: 'todosApi',
  baseQuery: fetchBaseQuery({ baseUrl: 'http://localhost:8080' }),
  endpoints: (builder) => {
    return {

      getTodos: builder.query({query: (id) => `/todos/detail/${id}`}),
    }
  }
})

export default todosApi;
```

src\toolkit\configureStore.js

```
import { combineReducers,applyMiddleware,createStore,compose } from 'redux';
import thunk from 'redux-thunk';
function isPlainObject(value) {
    if (typeof value !== "object" || value
        return false;
    return Object.getPrototypeOf(value)
}
function configureStore(options = {}) {
    let { reducer, middleware=[thunk], preloadedState } = options;
    let rootReducer;
    if (typeof reducer
        rootReducer = reducer;
    } else if (isPlainObject(reducer)) {
        rootReducer = combineReducers(reducer);
    }
+   middleware=typeof middleware === 'function'?middleware(()=>[thunk]):middleware
    const enhancer = applyMiddleware(...middleware);
    const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
    return createStore(rootReducer, preloadedState, composeEnhancers(enhancer));
}
export default configureStore;
```

src\toolkit\query\react.js

```
import { createSlice } from '../'
import { useEffect, useContext, useReducer } from 'react';
import { ReactReduxContext } from 'react-redux';
const FETCH_DATA = 'FETCH_DATA';
function fetchBaseQuery({ baseUrl }) {
    return async function (url) {
        url = baseUrl + url;
        let data = await fetch(url).then(res => res.json());
        return data;
    }
}

function createApi({ reducerPath, baseQuery, endpoints }) {
    let builder = {
        query(options) {
            function useQuery(id) {
                const { store } = useContext(ReactReduxContext)
                const [, forceUpdate] = useReducer(x => x + 1, 0);
                useEffect(() => {
                    let url = options.query(id);
                    store.dispatch({ type: FETCH_DATA, payload: { url } });
                    return store.subscribe(forceUpdate);
                }, [id, store])
                let state = store.getState();
                return state ? state[reducerPath] : {};
            }
            return { useQuery };
        }
    }
    let slice = createSlice({
        name: reducerPath,
        initialState: { data: null, error: null, isLoading: false },
        reducers: {
            setValue(state, { payload = {} }) {
                for (let key in payload)
                    state[key] = payload[key];
            }
        }
    });
    const { actions, reducer } = slice
    let api = {
        reducerPath,
        endpoints: endpoints(builder),
        reducer,
        middleware: function ({ dispatch }) {
            return function (next) {
                return function (action) {
                    if (action.type === FETCH_DATA) {
                        let { url } = action.payload;
                        ; (async function () {
                            try {
                                dispatch(actions.setValue({ isLoading: true }));
                                let data = await baseQuery(url);
                                dispatch(actions.setValue({ data, isLoading: false }));
                            } catch (error) {
                                console.log(error);
                                console.log(typeof error);
                                dispatch(actions.setValue({ error: { error: error.toString() }, isLoading: false }));
                            }
                        })();
                    } else {
                        next(action);
                    }
                }
            }
        }
    }
    return api;
}

export { fetchBaseQuery, createApi }
```

12.axios-basequery

src\todos.js

```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react'
import axios from 'axios'
axios.interceptors.response.use(function (response) {
  return {data:response.data};
},function (error){
  return {error:{error:error.message}};
});
const axiosBaseQuery = ({ baseUrl }) => (
  async (url) => {
    try {
      const result = await axios({ url: baseUrl + url })
      return result;
    } catch (error) {
      return error;
    }
  }
)

const todosApi = createApi({
  reducerPath: 'todosApi',
  baseQuery: axiosBaseQuery({ baseUrl: 'http://localhost:8080' }),
  endpoints: (builder) => {
    return {

      getTodos: builder.query({query: (id) => `/todos/detail/${id}`}),
    }
  }
})

export default todosApi;
```

参考