如下步骤感受Reducer的重构

**初遇 Reducer**

最初的reducer看起来如下，如果去重构呢？

const initialState = {

    visibilityFilter : 'SHOW\_ALL',

    todos : []

};

function appReducer(state = initialState, action) {

    switch(action.type) {

        case 'SET\_VISIBILITY\_FILTER' : {

            return Object.assign({}, state, {

                visibilityFilter : action.filter

            });

        }

        case 'ADD\_TODO' : {

            return Object.assign({}, state, {

                todos : state.todos.concat({

                    id: action.id,

                    text: action.text,

                    completed: false

                })

            });

        }

        case 'TOGGLE\_TODO' : {

            return Object.assign({}, state, {

                todos : state.todos.map(todo => {

                    if (todo.id !== action.id) {

                      return todo;

                    }

                    return Object.assign({}, todo, {

                        completed : !todo.completed

                    })

                  })

            });

        }

        case 'EDIT\_TODO' : {

            return Object.assign({}, state, {

                todos : state.todos.map(todo => {

                    if (todo.id !== action.id) {

                      return todo;

                    }

                    return Object.assign({}, todo, {

                        text : action.text

                    })

                  })

            });

        }

        default : return state;

    }

}

**提取工具函数**

工具函数也叫通用函数，如下我们提取了通用函数

// 更新对象

function updateObject(oldObject, newValues) {

    // 用空对象作为第一个参数传递给 Object.assign，以确保是复制数据，而不是去改变原来的数据

    return Object.assign({}, oldObject, newValues);

}

然后我们的修改我们的reducer，如下Add\_TODO的处理流程

        case 'ADD\_TODO' : {

            const newTodos = state.todos.concat({

                id: action.id,

                text: action.text,

                completed: false

            });

// todos 发送了改变，所以我们必须更新 todos

            return updateObject(state, {todos : newTodos});

        }

**提取 case reducer**

就是将case内的逻辑放到函数中（学过重构对这步应该比较熟悉吧），如下，我们将ADD\_TODO的代码分离为一个新的方法

function addTodo(state, action) {

    const newTodos = state.todos.concat({

        id: action.id,

        text: action.text,

        completed: false

    });

    return updateObject(state, {todos : newTodos});

}

然后修改我们的reducer

        case 'ADD\_TODO' :

            return addTodo(state, action);

**拆分reducer**

经过前面的重构，我们的reducer看起来如下：

function appReducer(state = initialState, action) {

    switch(action.type) {

        case 'SET\_VISIBILITY\_FILTER' :

            return setVisibilityFilter(state, action);

        case 'ADD\_TODO' :

            return addTodo(state, action);

        case 'TOGGLE\_TODO' :

            return toggleTodo(state, action);

        case 'EDIT\_TODO' :

            return editTodo(state, action);

        default : return state;

    }

}

现在把它拆分为3个reducer

function visibilityReducer(visibilityState = 'SHOW\_ALL', action) {

    switch(action.type) {

        case 'SET\_VISIBILITY\_FILTER' :

            return setVisibilityFilter(visibilityState, action);

        default :

            return visibilityState;

    }

};

function todosReducer(todosState = [], action) {

    switch(action.type) {

        case 'ADD\_TODO' :

            return addTodo(todosState, action);

        case 'TOGGLE\_TODO' :

            return toggleTodo(todosState, action);

        case 'EDIT\_TODO' :

            return editTodo(todosState, action);

        default : return todosState;

    }

}

// 根 reducer

function appReducer(state = initialState, action) {

    return {

        todos : todosReducer(state.todos, action),

        visibilityFilter : visibilityReducer(state.visibilityFilter, action)

    };

}

**通过切片组合 Reducer**

使用 Redux 中 combineReducers 这个工具函数去把管理每个 state 切片的逻辑组合起来

即，我们的根reducer如下：

// 根 reducer

function appReducer(state = initialState, action) {

    return {

        todos : todosReducer(state.todos, action),

        visibilityFilter : visibilityReducer(state.visibilityFilter, action)

    };

}

使用combineReducers 代替自己组合

// 顶层 reducer

const appReducer = combineReducers({

    visibilityFilter : visibilityReducer,

    todos : todosReducer

});

**最终重构后的reducer**

// 可重用的工具函数

function updateObject(oldObject, newValues) {

    // 将空对象作为第一个参数传递给 Object.assign，以确保只是复制数据，而不是去改变数据

    return Object.assign({}, oldObject, newValues);

}

// 可重用的工具函数

function updateItemInArray(array, itemId, updateItemCallback) {

    const updatedItems = array.map(item => {

        if(item.id !== itemId) {

            // 因为我们只想更新一个项目，所以保留所有的其他项目

            return item;

        }

         // 使用提供的回调来创建新的项目

        const updatedItem = updateItemCallback(item);

        return updatedItem;

});

    return updatedItems;

}

// SET\_VISIBILITY\_FILTER Case 处理方法

function setVisibilityFilter(visibilityState, action) {

    // 从技术上将，我们甚至不关心之前的状态

    return action.filter;

}

// visibility Reducer

function visibilityReducer(visibilityState = 'SHOW\_ALL', action) {

    switch(action.type) {

        case 'SET\_VISIBILITY\_FILTER' :

            return setVisibilityFilter(visibilityState, action);

        default :

            return visibilityState;

    }

};

// ADD\_TODO Case 处理方法

function addTodo(todosState, action) {

    const newTodos = todosState.concat({

        id: action.id,

        text: action.text,

        completed: false

});

    return newTodos;

}

// TOGGLE\_TODO Case 处理方法

function toggleTodo(todosState, action) {

    const newTodos = updateItemInArray(todosState, action.id, todo => {

        return updateObject(todo, {completed : !todo.completed});

});

    return newTodos;

}

// EDIT\_TODO Case 处理方法

function editTodo(todosState, action) {

    const newTodos = updateItemInArray(todosState, action.id, todo => {

        return updateObject(todo, {text : action.text});

});

    return newTodos;

}

// todos Reducer

function todosReducer(todosState = [], action) {

    switch(action.type) {

        case 'ADD\_TODO' : return addTodo(todosState, action);

        case 'TOGGLE\_TODO' : return toggleTodo(todosState, action);

        case 'EDIT\_TODO' : return editTodo(todosState, action);

        default : return todosState;

    }

}

// 顶层 reducer

const appReducer = combineReducers({

    visibilityFilter : visibilityReducer,

    todos : todosReducer

});