一行代码搞定一个reducer

——redux使用经验分享

效率产品组 小峰

不同的 redux 使用方式

1. 直接使用 redux
2. 使用 redux + @reduxjs/toolkit
3. 使用 redux + @reduxjs/toolkit + redux-use（推荐）

涉及技术：react、react hooks、redux、typescirpt

直接使用 redux （以常用的请求数据为例，示例代码使用 typescript 和 react hooks）

1. 定义 action type，编写 action creator
2. 编写 reducer
3. 视图中，使用 dispatch 派发 action

```tsx
/* eslint-disable react-hooks/exhaustive-deps */
import { useEffect } from 'react'
import { useDispatch } from 'react-redux'
import { load } from './reducer'

export default function Todo() {
    const dispatch = useDispatch()
    useEffect(() => {
        dispatch(load())
    }, [])

    return <div>Todos</div>
}
```

dispatch

```tsx
const LOAD = 'load'
const LOAD_SUCCESS = 'success'

const initState = {
    loading: false,
    loaded: false,
    data: []
}

export default function reducer(state = initState, action: any) {
    switch (action.type) {
        case LOAD: {
            return {
                ...state,
                loading: true
            }
        }
        case LOAD_SUCCESS: {
            return {
                ...state,
                loading: false,
                loaded: true,
                data: action.data
            }
        }
        default: return state
    }
}

export function load() {
    return async (dispatch: Dispatch) => {
        dispatch({type: LOAD})
        const data = await request<any>('GET', 'http://localhost:4000/todo')
        dispatch({type: LOAD_SUCCESS, data})
    }
}
```

action types

reducer

action creator

使用 redux + @reduxjs/toolkit

1. ~~定义 action type，编写 action creator~~
2. 编写 reducer
3. 视图中，使用 dispatch 派发 action

```
1  import { createSlice } from '@reduxjs/toolkit'
2
3  const initState = {
4      loading: false,
5      loaded: false,
6      data: []
7  }
8
9  const todoSlice = createSlice({
10     name: 'todo',
11     initialState: initState,
12     reducers: {
13         load: (state) => {
14             state.loading = true
15         },
16         success: (state, action) => {
17             state.loading = false
18             state.loaded = true
19             state.data = action.payload
20         }
21     }
22 })
23
24 export default todoSlice.reducer
25 export const actions = todoSlice.actions
```

reducers

```
1  /* eslint-disable react-hooks/exhaustive-deps */
2  import { useEffect } from 'react'
3  import { useDispatch } from 'react-redux'
4  import { actions } from './reducer2'
5  import request from './request'
6
7  export default function Todo() {
8      const dispatch = useDispatch()
9      useEffect(() => {
10         dispatch(actions.load())
11         request<any>('GET', 'http://localhost:4000/todo').then((data) => {
12             dispatch(actions.success(data))
13         })
14     }, [])
15
16     return <div>Todos</div>
17 }
18
19
```

dispatch

使用 redux + @reduxjs/toolkit + redux-use

1. ~~定义 action type，编写 action creator~~
2. ~~编写 reducer~~
3. ~~视图中，使用 dispatch 派发 action~~ 使用类似 useState 的写法获取数据 + 派发

```ts
import reduxu from 'redux-use'
import request from './request'

const createRequest = <P, R>(url: string, method: string = 'GET') => {
    const realUrl = url.startsWith('http') ? url : `http://localhost:4000/${url}`
    return async (params: P) => {
        return request<R, P>(method, realUrl, params)
    }
}

export const useTodo = reduxu.async(createRequest('./todo')).hook
export default reduxu.reducer()

```

传入请求函数

一行代码搞定

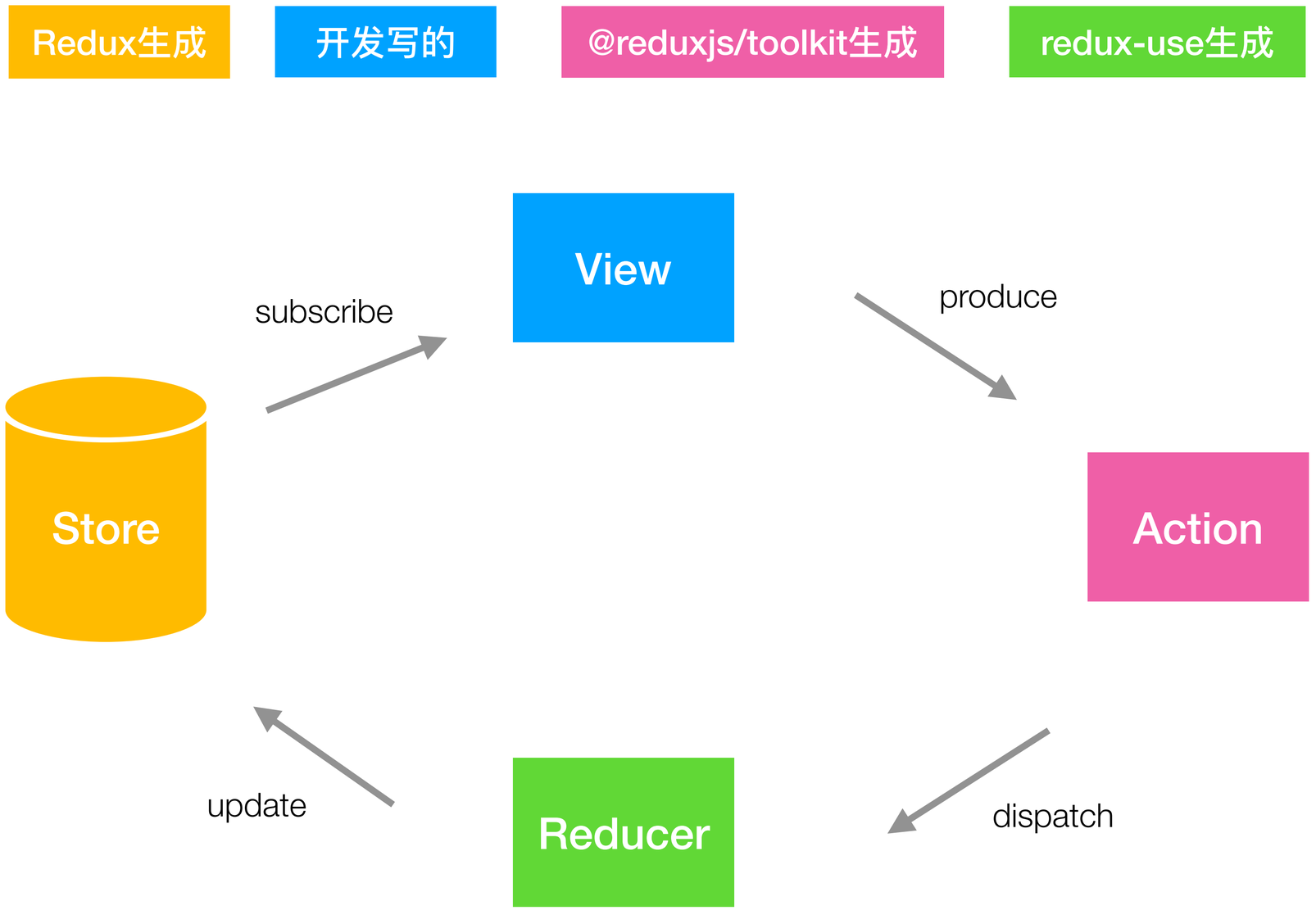
```ts
/* eslint-disable react-hooks/exhaustive-deps */
import { useEffect } from 'react'
import { useTodo } from './reducer3'

export default function Todo() {
    const [todo, loadTodo] = useTodo()
    useEffect(() => {
        loadTodo()
    }, [])

    return <div>Todos</div>
}
```

类似 useState 用法

最终效果


Redux生成
开发写的
@reduxjs/toolkit生成
redux-use生成
View
subscribe
produce
Store
Action
update
Reducer
dispatch

实际项目使用效果

1. 一项数据的增删改查

2. 修改、删除成功后，同步维护列表数据

```ts
import reduxu from 'redux-use'
import { create } from 'service'
import { createUrls, createUpdateToList, createRemoveFromList } from './helper'

const urls = createUrls('group')

type ListData = API.GroupController.ListData
type SaveParams = API.GroupController.SaveParams
type SaveData = API.GroupController.SaveData
type RemoveParams = API.GroupController.RemoveParams
type RemoveData = API.GroupController.RemoveData

const save = reduxu.async(
  create<SaveData, SaveParams>(urls.save, 'post')
)
export const useGroupSave = save.hook

const remove = reduxu.async(
  create<RemoveData, RemoveParams>(urls.remove, 'post')
)
export const useGroupRemove = remove.hook

export const useGroupList = reduxu.async(create<ListData>(urls.list), {
  extraReducers: (builder) => {
    builder.addCase(save.thunk.fulfilled, createUpdateToList<ListData, SaveData>('groups'))
      .addCase(remove.thunk.fulfilled, createRemoveFromList<ListData, RemoveData>('groups'))
  }
}).hook
```

请求参数、结果类型

保存

删除

请求列表

保存、删除成功更新列表

redux + @reduxjs/toolkit + redux-use 优势

1. 内置了常用的中间件，如 redux-thunk、immer (@reduxjs/toolkit)

2. 无需手动编写 reducer、action，减少工作量

3. 使用数据方式和 hooks 保持一致

4. typescript 友好，@reduxjs/toolkit、redux-use 可以很好地处理类型

5. api 简单，redux-use 只有 3 个 api