ToodList

1 使用Redux(不配合react-redux)

1.1 安装redux

```
npm i redux -S
```

1.2 创建store

创建`src/store/index.js`

```
// 引入
import { createStore } from 'redux'
// 引入对数据的操作
import reducer from '../reducer/index'


// 创建store
const store = createStore(
  reducer,
  // 加上
  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()
)

export default store
```

1.3 创建reducer

`src/reducer/index.js`

```
// 默认数据 空对象
const defaultState = {
  inputValue: '123',
  list: [
    'Racing car sprays burning fuel into crowd.',
    'Japanese princess to wed commoner.',
    'Australian walks 100km after outback crash.',
    'Man charged over missing wedding girl.',
    'Los Angeles battles huge wildfires.'
  ]
}

// 存储如何对数据进行操作
/**
 * state 存储的数据 [ 上一次传递过去的数据 ]
```

```
 */
export default (state = defaultState, action) => {
  // 接受action 处理数据
  if (action.type === 'change_input_value') {
    // 不能直接修改state
    const newState = JSON.parse(JSON.stringify(state))
    // 会拿新的state数据，去替换旧的state数据
    newState.inputValue = action.inputValue
    return newState
  }
  if (action.type === 'add_item') {
    const newState = JSON.parse(JSON.stringify(state))
    newState.list.push(action.inputValue)
    return newState
  }
  if (action.type === 'delete_item') {
    const newState = JSON.parse(JSON.stringify(state))
    newState.list.splice(action.index, 1)
    return newState
  }
  return state
}
```

1.4 在List种使用

```
import React, { Component } from 'react'
import { Button, Input, List } from 'antd'
import './ToodList.scss'
import store from './store'


class ToodList extends Component {
  constructor () {
    super()
    // 获取store 赋值给store
    this.state = store.getState()
    this.handleStoreChange = this.handleStoreChange.bind(this)
    // 订阅store 当store发生改变后，执行handleStoreChange
    store.subscribe(this.handleStoreChange)
  }
  // 输入框值发生改变
  handleValueChange (e) {
    store.dispatch({
      type: 'change_input_value',
      inputValue: e.target.value
    })
  }
  // 向列表中添加项目
  handleAddList () {
    store.dispatch({
      type: 'add_item',
      inputValue: this.state.inputValue
```

```
    })
  }
  // 删除列表中的项 传递index
  handleDeleteItem (index) {
    store.dispatch({
      type: 'delete_item',
      index
    })
  }
  // 当store发生变化时
  handleStoreChange () {
    console.log('store change')
    // 同步数据
    this.setState(store.getState())
  }
  render () {
    return (
      <div className="ToodList aa">
        <Input style={{ width: 240, marginRight: 20 }}
               value={ this.state.inputValue }
               onChange={ e=> this.handleValueChange(e) }
        />
        <Button type="primary" onClick={ e=> this.handleAddList() }>添加</Button>
        <List
          style={{ width: 240, marginTop: 20 }}
          size="small"
          bordered
          dataSource={this.state.list}
          renderItem={(item, index) =>
            <List.Item onClick={ e=> this.handleDeleteItem(index) }>{item}</List.Item>
          }
        />
      </div>
    )
  }
}

export default ToodList
```

2 拆分Action Type

把Type变量抽出来作为常量`src/action/index.js`

```
export const CHANGE_INPUT_VALUE = 'change_input_value'

export const ADD_ITEM = 'add_item'

export const DELETE_ITEM = 'delete_item'
```

把Action抽出来，使用函数来创建`src/actionCreators/index.js`

```
import { CHANGE_INPUT_VALUE, ADD_ITEM, DELETE_ITEM  } from '../actionTypes'

export const getInputChangeAction = (value) => ({
  type: CHANGE_INPUT_VALUE,
  value
})
```

3 Redux 设计和使用的三项原则

- store是唯一的
- 只有store改变自己的内容

> reducer只是拿到以前的state，然后复制一个副本，再对这个副本做数据的处理，然后把这个副本返回给store，让store自己更改自己的数据

- Reducer 必须是一个纯函数

> 给定固定的输入，就一定有固定的输出，并且不会有任何副作用

3.1 核心API

- createStore
- store.dispatch
- store.getState
- store.subscribe