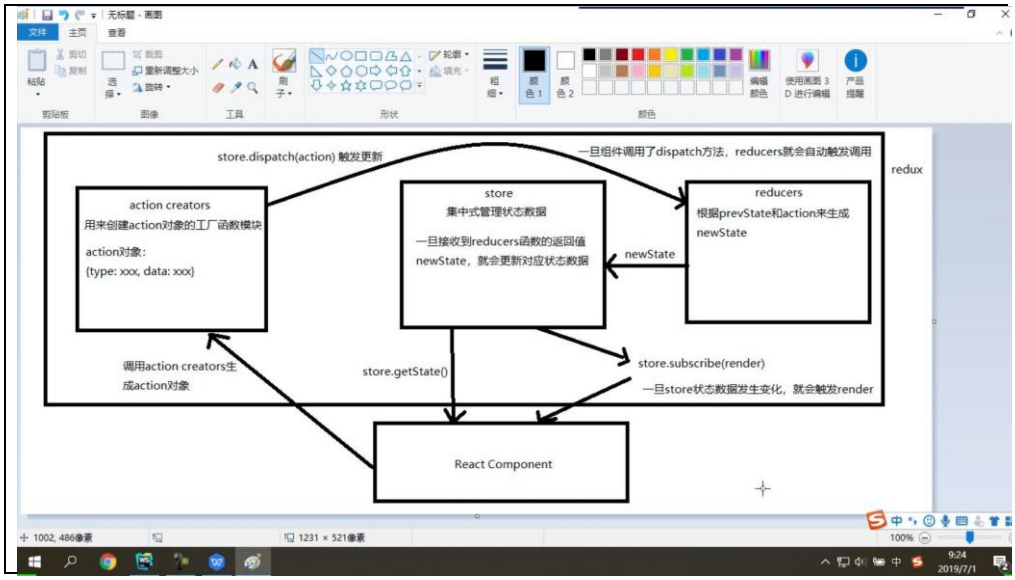
redux 个人笔记（最终版，无过程）

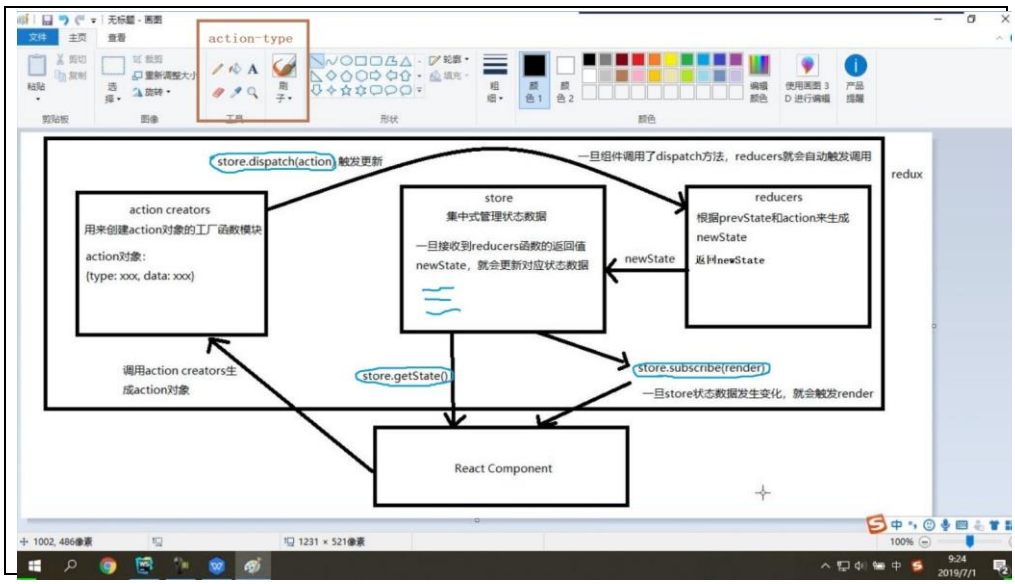
是什么？ 是一个状态管理器

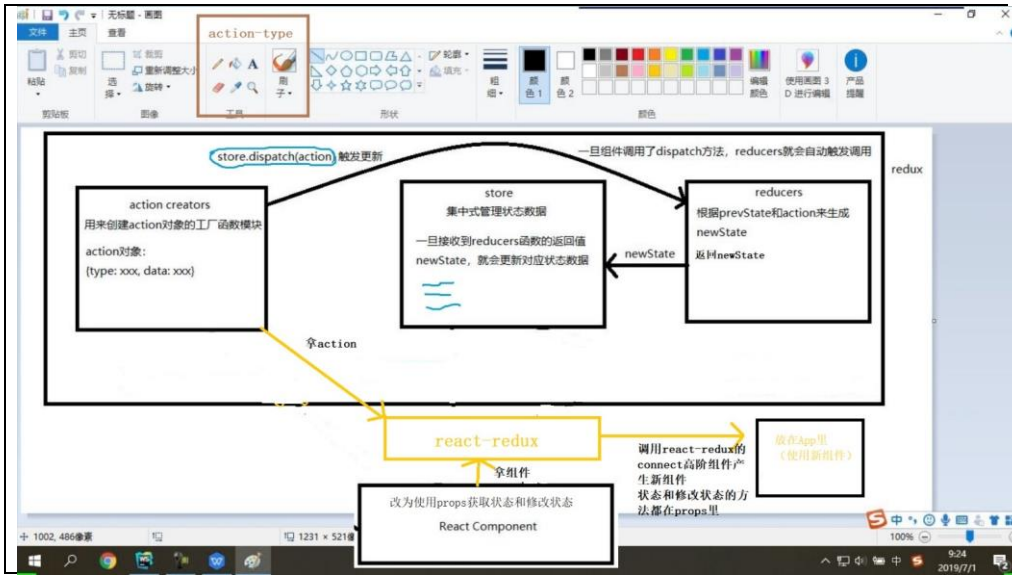
作用：集中性管理多个组件的共享状态

原则： 单一数据源（多个组件公用一个状态源）、数据只读（只有 getState、设置要通过 dispatch 间接完成）、使用纯函数更新数据（同样的输入同样的输出、dispatch 和 dispatch 里的 reducers）

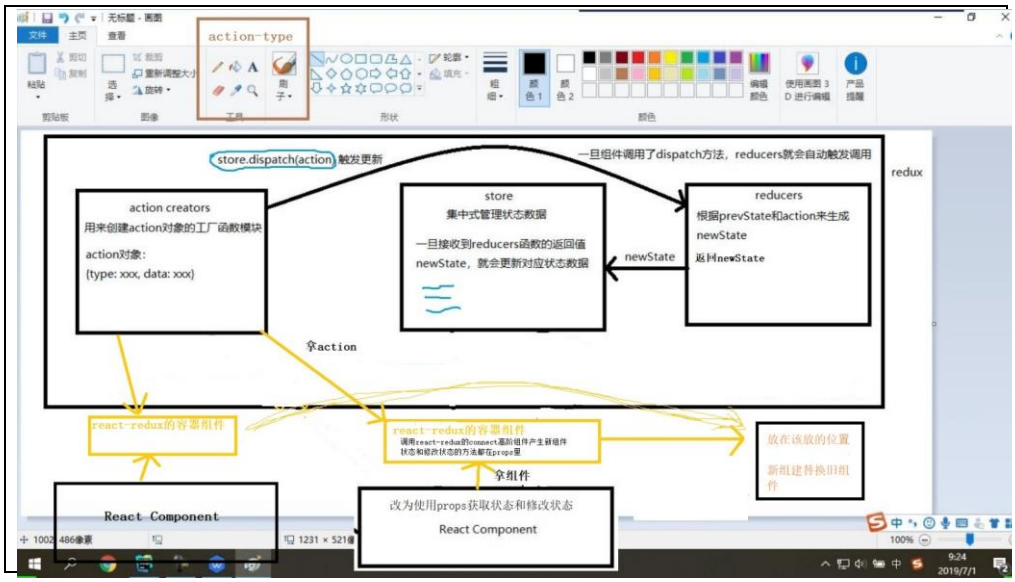
批注 [浪1]: 数据都是放在了 store 里面（所以是统一管理），



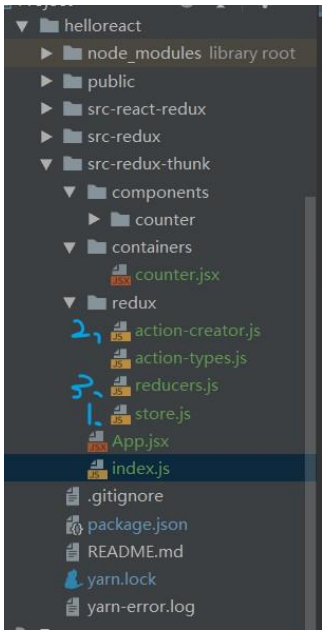




多个组件使用同一个 redux



1、上脚手架



index.js

```
/**
 * react-redux
 *   1.合并创建 action 和 dispatch 的代码
 *   2.本文件 Provider 代替 store.subscribe(render)
 *    自动检测状态变化，重新渲染
 * /

import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux'

import App from './App';
import store from './redux/store'

ReactDOM.render(<Provider store={store}><App /></Provider>, document.getElementById('root'));
```

App.jsx

```
import React,{Component} from 'react';

import Counter from './containers/counter'   // 使用 react-redux 的容器组件
```

```
4.
5.  export default class App extends Component{
6.    render() {
7.      return <Counter/>
8.    }
9.  }
```

普通组件 Counter

```
1.  import React,{ Component, Fragment } from 'react';
2.  import PropTypes from 'prop-types'
3.
4.  /**
5.   * 使用了 react-redux 的容器组件后
6.   *  1. 状态 num 在 props 上
7.   *  2. 创建 action 和 dispatch 合并在一起，合并后的方法也在 props 上
8.   *  （合并后直接使用 ：this.props.increment(this.state.value);）
9.   * /
10.
11. export default class Counter extends Component{
12.   static propTypes = {
13.     num: PropTypes.number.isRequired,            // 接受状态数据
14.     increment: PropTypes.func.isRequired,       // 接受 action 方法
15.     decrement: PropTypes.func.isRequired,
16.   };
17.
18.   state = {
19.     value: 1
20.   };
21.
22.   handleChange = (e) => {
23.     this.setState({
24.       value: +e.target.value
25.     });
26.   };
27.
28.   increment = () => {
29.     this.props.increment(this.state.value);
30.   };
31.
32.   incrementAsync = () => {
33.     this.props.incrementAsync(this.state.value);
34.   };
35.
36.   render() {
37.     const {num} = this.props; // 在 props 拿状态 num
38.
39.     return <Fragment>
```

批注 [浪2]: |
在 react 里面，而不是 antd ui

```
      <h2>click {num} times</h2>
      <select onChange={this.handleChange}>
        <option value="1">1</option>
        <option value="2">2</option>
        <option value="3">3</option>
      </select>
      <button onClick={this.increment}>+</button>
      <button onClick={this.incrementAsync}>increment async</button>
    </Fragment>;
  }
}
```

Store.js

```
/**
 * 创建 store 对象
 * const store = createStore(reducers, composeWithDevTools(applyMiddleware(thunk)));
 * 参数：功能实现 reducers、异步 thunk
 * /

import { createStore, applyMiddleware } from 'redux'
import { composeWithDevTools } from 'redux-devtools-extension'

import reducers from './reducers'
import thunk from 'redux-thunk'

/**
 * redux-thunk
 *   1.执行异步代码
 *      thunk 放在 createStore 第二个参数，用应用中间件 applyMiddleware 包裹
 *
 * redux-devtools-extension
 *   1.异步代码检测（上线需要删除改组件）
 * /

const store = createStore(reducers, composeWithDevTools(applyMiddleware(thunk)));
export default store;
```

批注 [浪3]: 创建 store 是传 reducers 参数:

const store = createStore（reducers）

调用 store.dispatch 时传参 action:

这几个文件中没有，因为使用了 react-redux，都在容器里面封装了，通过 props 传给要用的组件

action-creator：（创建 action）(间接修改数据、异步修改数据放这)

```
import { INCREMENT, DECREMENT, ERROR } from './action-types'
// 根据 state 的功能来定义 action，目前只有加减的操作

// 同步 返回 action 对象
export const increment = (value) => ({ type:INCREMENT, data:value });
export const decrement = (value) => ({ type:DECREMENT, data:value });
```

批注 [浪4]: action-types：

```
export const INCREMENT = 'INCREMENT';
export const DECREMENT = 'DECREMENT';
export const ERROR = 'ERROR';
```

```
export const error = (value) => ({ type:ERROR, data:value });

// 异步 返回函数
export const incrementAsync = (value) => {
  return (dispatch) => {
    setTimeout(() => {
      // 模拟发送请求
      // 成功
      dispatch(increment(value));
      // 失败
      // dispatch(error('请求失败'))
    },1000)
  }
};
```

reducers.js（返回 newState）（直接修改数据放这）

```
/**
 * 根据 action.type 实现 state 的具体操作功能
 *   不存在的 action.type 返回匹配 default
 * /
import {INCREMENT, DECREMENT, ERROR} from './action-types'; // 引入 action-types 用于判断 action.type
import { combineReducers } from 'redux';

function num(prevState = 0, action) {
  switch (action.type) {          // 判断 action 的类型，(action 的类型是引 action-types 的， 这里也引)
    case INCREMENT :
      return prevState + action.data;
    case DECREMENT :
      return prevState - action.data;
    case ERROR:
      console.log(action.data);
      return prevState;
    default :
      return prevState;
  }
}

 function add(prevState = [], action){}

export default num; // 暴露 num 这个 reducer 函数

// 暴露多个 （当组件仅需要获取某个状态的时候 使用 结构赋值获取，const {num} = this.props.getState())
// export default combineReducers（{ // 暴露的还是 reducer 函数
//   num:
//   add;
//   yyy
// }）
```

```
32.
```

容器组件（将 React 与 redux 关联起来）（多个 UI 组件共用 state 就创建多个容器组件，传对应的普通(UI)组件）

```
1.  import { connect } from 'react-redux';
2.
3.  import Counter from '../components/counter';
4.  import { increment, decrement, incrementAsync, error } from '../redux/action-creator'  // 导入各
    个 action 工厂函数
5.
6.  /**
7.   * react-redux
8.   *    1.高阶组件 connect 封装 action 和 dispatch 到一起，传到 Counter 普通组件 props 中使用
9.   * /
10. export default connect(
11.   (state) => ({num: state }),                    // 暴露一个
12.   { increment, decrement, incrementAsync, error }
13. )(Counter)              // 注意：这里不是传组件
```

```
// 暴露多个
 // (state) => ({num: state.num, add: state.add})
```

批注 [浪5]: 原生不用 react-redux 时，通过父组件传 store 给要用的组件