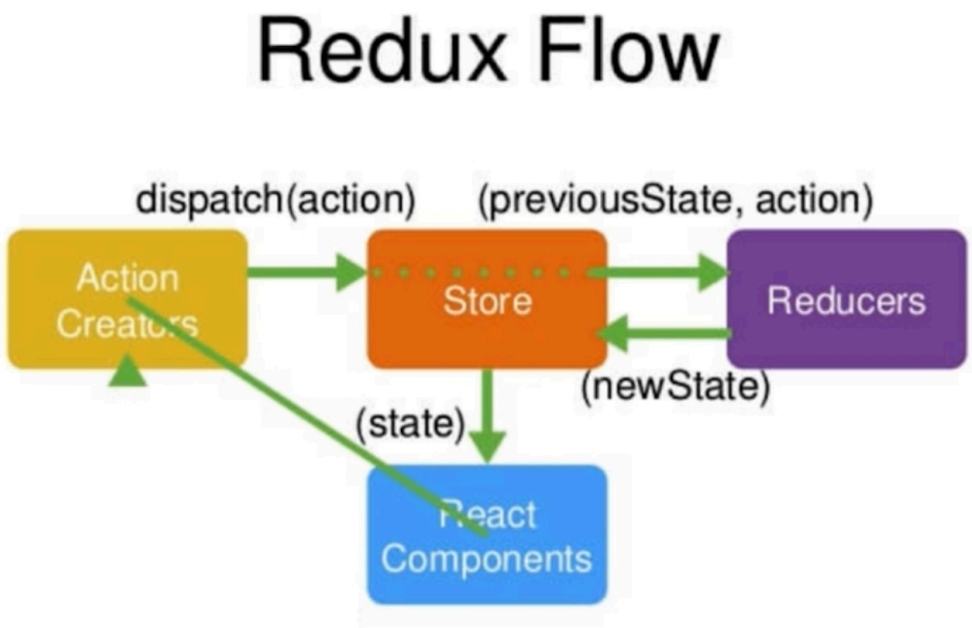
React细节知识点 - Redux篇

1. Redux Flow



——

2. 使用Antd实现布局

1. 使用

```
//安装
yarn add antd
//导入
import 'antd/dist/antd.css';
//需要用到什么组件就导入什么组件
import { Input, Button, List } from 'antd';

const data = [
  '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.'
];
```

```
<div style={{ marginTop: '10px', marginLeft: '10px' }}>
  <div>
    <Input
      placeholder="todo info"
      style={{ width: '300px', marginRight: '10px' }}
      />
    <Button type="primary">提交</Button>
  </div>
  <List
    style={{ marginTop: '10px', width: '300px' }}
    bordered
    dataSource={data}
    renderItem={item => <List.Item>{item}</List.Item>}
    />
</div>
```

---

3. 创建redux中的store

1. **安装 & 创建store & reducer**

```
//1. yarn add redux
//2. 在src目录下新建store文件夹
//3. 在store目录下新建index.js，创建store的代码就放在index.js中

import { createStore } from 'redux';

const store = createStore();

export default store;

//4. 在store目录下新建reducer.js
const defaultState = {
  inputValue: '',
  list: []
}

export default (state = defaultState, action) => {      //state:整个仓库存储的数据
  return state;
}
```

```
//5. 把笔记本(reducer.js)引入到store(index.js)中
index.js:
import { createStore } from 'redux';
import reducer from './reducer';

const store = createStore(reducer);

export default store;

//6. 接着我们想在组件里调用store这个仓库里的数据
import store from './store';

...
constructor(props) {
  super(props);
  //console.log(store.getState()); // 打印出了我们在store里设置的defaultState
  this.state = store.getState();   //这样我们就把store里存储的数据放入了组件的state
里
}                                  // 下面组件模板里用的时候直接{this.state.list}
```

——

4. Action 和 Reducer的编写

1. **使用redux devtools**

```
//在store/index.js中加入：window.....这行代码
const store = createStore(
  reducer,
  window.__REDUX_DEVTOOLS_EXTENSION__ &&
window.__REDUX_DEVTOOLS_EXTENSION__()
);

export default store;
```

2. **创建、派发action —> reducer接收、操作 —> store订阅获取reducer操作完的数据 —> 组件通过getState()获取更新后store里的数据**

```
//1. <Input>标签上绑定onChange={this.handleInputChange}事件
```

```
//2. 事件里创建action并派发action
  handleInputChange = e => {
    const action = {                    //创建action
      type: 'change_input_value',
      value: e.target.value
    };
    store.dispatch(action);             //派发action
  };

//3. reducer.js中接收action并根据type做出对应的操作
// reducer 可以接收state，但是绝不能修改state，这是为什么我们先进行深拷贝
  export default (state = defaultState, action) => {
    if (action.type === 'change_input_value') {
      //先对原本的state进行一次深拷贝
      const newState = JSON.parse(JSON.stringify(state));
      newState.inputValue = action.value;
      return newState;
    }
    return state;
  };

//4. store订阅获取reducer操作完的数据
  constructor(props) {
    super(props);
    this.state = store.getState();
    this.handleInputChange = this.handleInputChange.bind(this);
    this.handleStoreChange = this.handleStoreChange.bind(this);    -->新增
    store.subscribe(this.handleStoreChange);                       -->新增
  }

//5. 组件通过getState()获取store里更新过后的数据
  handleStoreChange = () => {
    this.setState(store.getState());
  };
```

——

5. ActionTypes的拆分

1. **操作**

```
//1.在store目录下新建actionTypes.js
```

```
//2.将action里的type写成常量形式
export const CHANGE_INPUT_VALUE = 'change_input_value';
export const ADD_TODO_ITEM = 'add_todo_item';
export const DELETE_TODO_ITEM = 'delete_todo_item';

//3.其他文件里需要用到则导入
TodoList.js文件
import {
  CHANGE_INPUT_VALUE,
  ADD_TODO_ITEM,
  DELETE_TODO_ITEM
} from './store/actionTypes';


/*原本写法*/
  handleInputChange = e => {
    const action = {
      type: 'change_input_value',          --->这里
      value: e.target.value
    };
    store.dispatch(action);
  };


/*改变后写法*/
  handleInputChange = e => {
    const action = {
      type: CHANGE_INPUT_VALUE,            --->这里
      value: e.target.value
    };
    store.dispatch(action);
  };

reducer.js文件
import {
  CHANGE_INPUT_VALUE,
  ADD_TODO_ITEM,
  DELETE_TODO_ITEM
} from './actionTypes';

  if (action.type === CHANGE_INPUT_VALUE) {
    //先对原本的state进行一次深拷贝
    const newState = JSON.parse(JSON.stringify(state));
    newState.inputValue = action.value;
    return newState;
  }
```

---

6. 使用actionCreator统一创建action

1. 操作

```
//1.在store目录下新建actionCreators.js

//2.把创建action放在这个文件里
import {
  CHANGE_INPUT_VALUE,
  ADD_TODO_ITEM,
  DELETE_TODO_ITEM
} from './actionTypes';

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

export const getAddItemAction = () => ({
  type: ADD_TODO_ITEM
});

export const getDeleteItemAction = index => ({
  type: DELETE_TODO_ITEM,
  index
});

//3.在TodoList.js中导入使用
import {
  getInputChangeAction,
  getAddItemAction,
  getDeleteItemAction
} from './store/actionCreators';

/*原本写法*/
  handleInputChange = e => {
    const action = {
      type: CHANGE_INPUT_VALUE,
      value: e.target.value
    };
    store.dispatch(action);
```

```
  };

/*改变后写法*/
  handleInputChange = e => {
    const action = getInputChangeAction(e.target.value);
    store.dispatch(action);
  };

/*原本写法*/
  handleBtnClick = () => {
    const action = {
      type: ADD_TODO_ITEM
    };
    store.dispatch(action);
  };

/*改变后写法*/
  handleBtnClick = () => {
    const action = getAddItemAction();
    store.dispatch(action);
  };

/*原本写法*/
  handleItemDelete = index => {
    const action = {
      type: DELETE_TODO_ITEM,
      index
    };
    store.dispatch(action);
  };

/*改变后写法*/
  handleItemDelete = index => {
    const action = getDeleteItemAction(index);
    store.dispatch(action);
  };
```

---

7. 总结

1. **store是唯一的**

2. **只有store能够改变自己的内容；很多人会认为是reducer.js中的操作直接改变了store里的内容，其实是错误的。reducer是拿到之前store里的数据，它做了一次深拷贝，在拷贝体上做操作，然后把这个操作完的newState传回给store。store接收到reducer传来的数据，自己更新自己的store里的数据**
3. **Reducer必须是纯函数；纯函数指的是，给定固定的输入，就一定会有固定的输出，而且不会有任何副作用**
4. **Redux中的核心API**
    1. **createStore —> 帮助我们创建一个store**
    2. **store.dispatch —> 帮助我们派发action**
    3. **store.getState —> 帮助我们获取到store里的所有数据内容**
    4. **store.subscribe —> 可以让我们订阅store的改变，只要store发生改变，store.subscribe这个函数接收的回调函数就会被执行**