React细节知识点 - Redux 篇2

1. UI组件 | 容器组件

1. **UI组件：只负责页面的渲染**

```
//新建TodoListUI.js
import React, { Component } from 'react';
import { Input, Button, List } from 'antd';

class TodoListUI extends Component {
  render() {
    return (
      <div style={{ marginTop: '10px', marginLeft: '10px' }}>
        <div>
          <Input
            value={this.props.inputValue}
            placeholder="todo info"
            style={{ width: '300px', marginRight: '10px' }}
            onChange={this.props.handleInputChange}
          />
          <Button type="primary" onClick={this.props.handleBtnClick}>
            提交
          </Button>
        </div>
        <List
          style={{ marginTop: '10px', width: '300px' }}
          bordered
          dataSource={this.props.list}
          renderItem={(item, index) => (
            <List.Item onClick={() =>
              {this.props.handleItemDelete(index)}}>
              {item}
            </List.Item>
          )}
        />
      </div>
    );
  }
}

export default TodoListUI;
```

2. **页面渲染部分代码放到TodoListUI组件中，剩下的TodoList.js称为容器组件**

```
import TodoList from './TodoListUI';

class TodoList extends Component {
  constructor(props) {
    super(props);
    this.state = store.getState();
    this.handleInputChange = this.handleInputChange.bind(this);
    this.handleBtnClick = this.handleBtnClick.bind(this);
    this.handleItemDelete = this.handleItemDelete.bind(this);
    this.handleStoreChange = this.handleStoreChange.bind(this);
    store.subscribe(this.handleStoreChange);
  }

  render() {
    <TodoListUI       //父组件把state里的值和方法传给UI组件
      inputValue={this.state.inputValue}
      list={this.state.list}
      handleInputChange={this.handleInputChange}
      handleBtnClick={this.handleBtnClick}
      handleItemDelete={this.handleItemDelete}
    />
  }

  handleInputChange = e => {
    const action = getInputChangeAction(e.target.value);
    store.dispatch(action);
  };

  handleBtnClick = () => {
    const action = getAddItemAction();
    store.dispatch(action);
  };

  handleItemDelete = index => {
    const action = getDeleteItemAction(index);
    store.dispatch(action);
  };

  handleStoreChange = () => {
    this.setState(store.getState());
  };
}
```

```
export default TodoList;
```

——

2. 无状态组件

1. **当一个组件只有一个render函数的时候，我们就可以用一个无状态组件来定义这个组件，其实无状态组件就是一个函数**
2. **无状态组件的优势：性能比较高，因为它就是一个函数**

```
//我们上面写的TodoListUI.js其实组件里就是只有一个render函数，所以可以改写成无状态组件
//无状态组件有一个参数是props
const TodoListUI = (props) => {
  return (
    <div style={{ marginTop: '10px', marginLeft: '10px' }}>
        <div>
          <Input
            value={props.inputValue}
            placeholder="todo info"
            style={{ width: '300px', marginRight: '10px' }}
            onChange={props.handleInputChange}
          />
          <Button type="primary" onClick={this.props.handleBtnClick}>
            提交
          </Button>
        </div>
        <List
          style={{ marginTop: '10px', width: '300px' }}
          bordered
          dataSource={props.list}
          renderItem={(item, index) => (
            <List.Item onClick={(index) =>
              {props.handleItemDelete(index)}}>
              {item}
            </List.Item>
          )}
        />
     </div>
  )
}
```

---

3. Redux中发送异步请求获取数据

1. 操作

```
//1.安装yarn add axios

//2.写代码
/*TodoList.js*/
import axios from 'axios';

componentDidMount() {
  axios.get('/api/list.json').then(res => {    //请求获取到数据后，放入store中
    const data = res.data;
    const action = initListAction(data);
    store.dispatch(action);
  });
}

/*actionTypes.js中*/
export const INIT_LIST_ACTION = 'init_list_action';

/*actionCreators.js中*/
export const initListAction = data => ({
  type: INIT_LIST_ACTION,
  data
});

/*reducer.js中*/
if (action.type === INIT_LIST_ACTION) {
  const newState = JSON.parse(JSON.stringify(state));
  newState.list = action.data;
  return newState;
}
```

---

4. Redux-thunk中间件进行ajax请求发送

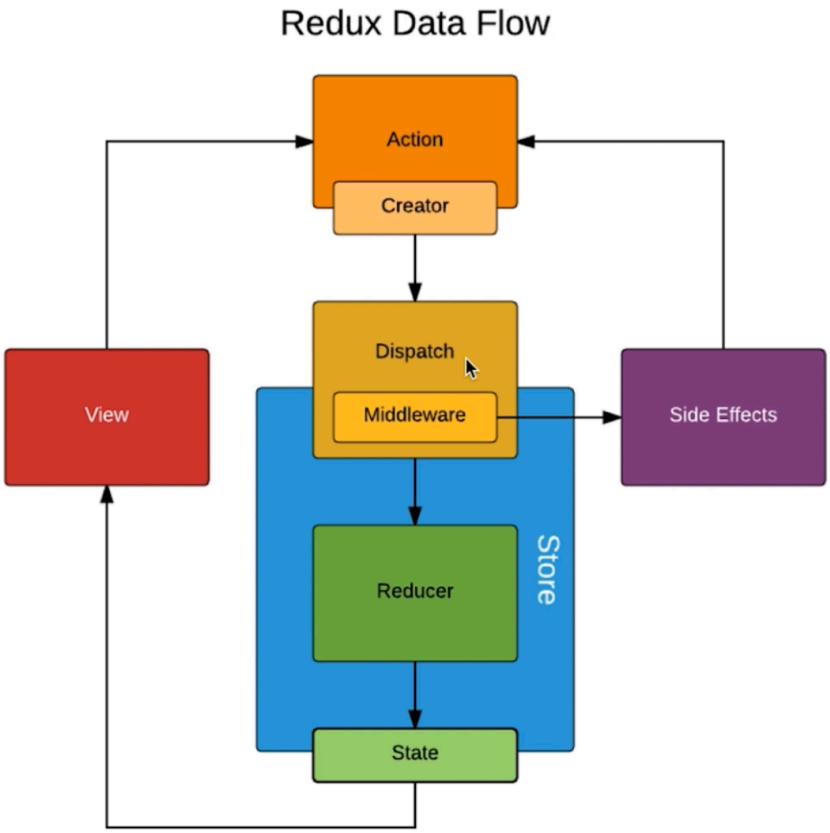
1. **如果我们把异步请求后者复杂的逻辑都放在组件里的话，组件会显得过于臃肿，所以遇到这种异步请求或者非常复杂的逻辑，我们希望把它移到其他地方进行统一的管理。Redux-thunk这个中间件可以使得我们将异步请求(复杂的逻辑)放在action中去处理；Redux-thunk是Redux的一个中间件**
2. **操作**

```
//1.安装
yarn add redux-thunk

//2.如何配置
import { createStore, applyMiddleware, compose } from 'redux';
import thunk from 'redux-thunk';

const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__
  ? window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__({})
  : compose;

const enhancer = composeEnhancers(
  applyMiddleware(thunk)
  // other store enhancers if any
);

const store = createStore(reducer, enhancer);

export default store;

//3.把异步请求操作代码放到actionCreators.js中
import axios from 'axios';

export const getTodoList = () => {    //这个函数返回的是一个函数对象
  return dispatch => {                ---> 注意这里直接可以有个dispatch
    axios.get('/api/list.json').then(res => {
      const data = res.data;          //获取到数据后，把数据放入store中，通过派发
action
      const action = initListAction(data);
      dispatch(action);
    });
  };
};

/*TodoList.js中*/
componentDidMount() {
  const action = getTodoList();    //因为写在actionCreators.js中的是返回一个函数对
象
  store.dispatch(action);      //所以当dispatch(action)时会去执行那个异步请求的函数
```

```
}                          //因此异步请求函数里的dispatch(action)是把获取到的数据放入store
中
```

---

5. 什么是Redux中间件



Redux中间件其实就是Action和Store中间，它其实就是对Store的Dispatch方法做一个升级，之前这个Dispatch方法只能接收一个对象，现在升级后 [使用redux-thunk进行升级] 可以接收对象也可以接收一个函数了，而我们的异步ajax请求操作返回的就一个函数，然后dispatch(这个函数action)

Redux-thunk 和 Redux-saga 区别：

1. **Redux-thunk把异步操作放在actionCreators里**
2. **Redux-saga把异步操作放在一个单独的文件里**

---

6. Redux-saga中间件

1. 操作

```
//1.安装
yarn add redux-sage

//2.配置 - 之前只有reducer能获取派发的action，使用了中间件，中间件也能获取到派发的actoon，然后我们就在这个sagas.js中写异步操作
/*store/index.js中，也就是创建store的地方*/
import reducer from './reducer';
import createSagaMiddleware from 'redux-saga';
import todoSagas from './sagas';

const sagaMiddleware = createSagaMiddleware();
const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__
  ? window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__({})
  : compose;

const enhancer = composeEnhancers(applyMiddleware(sagaMiddleware));

const store = createStore(reducer, enhancer);
sagaMiddleware.run(todoSagas);

export default store;

/*store/sagas.js*/
import { takeEvery, put } from 'redux-saga/effects';
import { GET_INIT_LIST } from './actionTypes';
import { initListAction } from './actionCreators';
import axios from 'axios';

function* getInitList() {
  try {
    const res = yield axios.get('/api/list.json');
    const action = initListAction(res.data);
    yield put(action);
  } catch (e) {
    console.log('list.json 网络请求失败');
  }
}

function* mySaga() {
  yield takeEvery(GET_INIT_LIST, getInitList);  //捕获类型是GET_INIT_LIST的action
}
```

```
export default mySaga;
```

——

7. React-redux的使用

react-redux是一个第三方模块，帮助我们在react中更方便的使用redux

1. **Redux 和 React-redux 写法对比**
    1. **Redux写法：(1) 先创建store和reducer (2) 在组件中【TodoList.js中】，在constructor函数中，通过this.state=store.getState()获取store里存储的数据**

```
/*Redux写法*/
//1.新建store文件夹，在其中新建index.js, reducer.js
/*store/index.js中*/
import { createStore } from 'redux';
import reducer from './reducer';

const store = createStore(reducer);

export default store;

/*store/reducer.js中*/
const defaultState = {
  inputValue: 'hello world',
  list: []
};

export default (state = defaultState, action) => {
  return state;
};

//2.在TodoList组件里
/*TodoList.js中*/
使用store里的数据是通过在constructor函数里写this.state=store.getState()
import React, { Component } from 'react';
import store from './store';

class TodoList extends Component {
  constructor(props) {
    super(props);
```

```
    this.state = store.getState();
  }
  render() {
    return (
      <div>
        <div>
          <input type="text" value={this.state.inputValue} />
          <button>提交</button>
        </div>
        <ul>
          <li>Chenxi</li>
        </ul>
      </div>
    );
  }
}
export default TodoList;
```

2. **React-redux写法**

```
//1.在根目录的index.js中
import { Provider } from 'react-redux';    //安装yarn add react-redux
import store from './store';

const App = (
  <Provider store={store}>    //Provider这个提供器连接了store,那么Provider
    <TodoList />              //里面的所有组件都有能力获取到store里的内容
  </Provider>
)

ReactDOM.render(App, document.getElementById('root'));

//2.在TodoList组件里，在根目录index.js中，TodoList在Provider提供器里，
//组件里只需再使用connect做链接store即可，有两个规则：
//(1)mapStateToProps，把store的数据传递给这个组件的props
//(2)
import React, { Component } from 'react';
import store from './store';
import { connect } from 'react-redux';      //改变了这里

class TodoList extends Component {
  render() {
    return (
      <div>
```

```
        <div>
          <input type="text" value={this.props.inputValue} />
          <button>提交</button>
        </div>
        <ul>
          <li>Chenxi</li>
        </ul>
      </div>
    );
  }
}

const mapStateToProps = (state) => {    //这里的state看成store
  return {
    inputValue: state.inputValue      //store里的inputValue放到props的
InputValue
  }
}

const mapDispatchToProps = dispatch => {
  return {
    changeInputValue(e) {
      const action = {
        type: 'change_input_value',
        value: e.target.value
      };
      dispatch(action);
    }
  };
};

export default connect(mapStateToProps, mapDispatchToProps)(TodoList);//改变了
这里
```