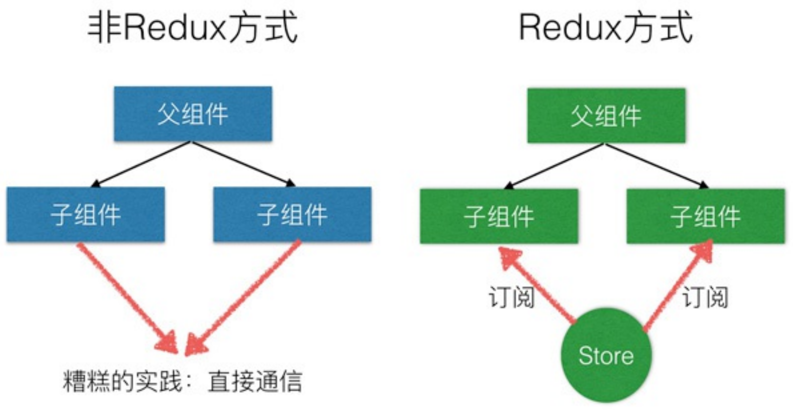
link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=47 sentences=143, words=1274

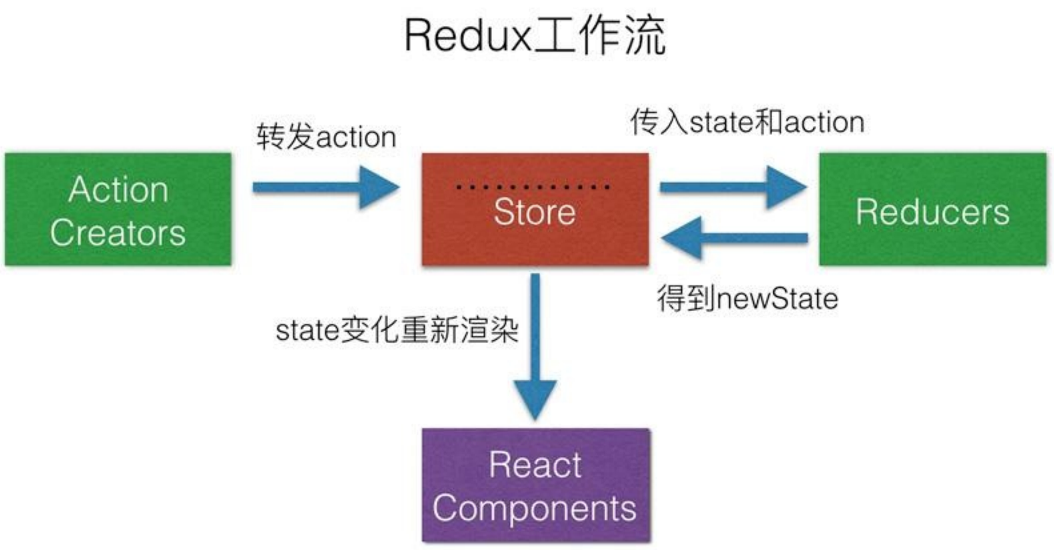
1.Redux应用场景

- 在React中，数据在组件中是单向流动的
- 数据通过props从父组件流向子组件
- 两个兄弟组件之间的通信就比较麻烦



2.Redux设计思想

- Redux是将整个应用状态存储到到一个地方，称为 `store`
- 里面保存一棵状态树 `state tree`
- 组件可以派发 `dispatch action`给 `store`,而不是直接通知其它组件
- 其它组件可以通过订阅 `store`中的状态(state)来刷新自己的视图



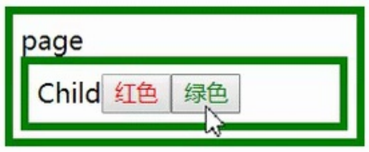
3.预备知识

3.1 redux全家桶

- [redux (https://github.com/reduxjs/redux)](https://github.com/reduxjs/redux)是 JavaScript 状态容器， 提供可预测化的状态管理
- [redux-logger (https://github.com/LogRocket/redux-logger)](https://github.com/LogRocket/redux-logger)可以打印状态变化前后的日志
- [redux-thunk (https://github.com/reduxjs/redux-thunk)](https://github.com/reduxjs/redux-thunk)可以让store可以dispatch函数
- [redux-promise (https://github.com/redux-utilities/redux-promise)](https://github.com/redux-utilities/redux-promise)可以让store可以派发promise
- [react-redux (https://github.com/reduxjs/react-redux)](https://github.com/reduxjs/react-redux)可以实现React组件和Redux的连接，让组件自动订阅仓库中的状态变化事件，并状态发生变化的时候自动更新

3.2 Context(上下文)

- 在某些场景下，你想在整个组件树中传递数据，但却不想手动地在每一层传递属性。你可以直接在 React 中使用强大的contextAPI解决上述问题



```
import React,{Component} from 'react';
import ReactDOM from 'react-dom';
let ThemeContext = React.createContext('theme');
class Child extends Component {
    render() {
        return (
             <ThemeContext.Consumer>
                {
                    value => (
                        <div style={{ border: `5px solid ${value.color}`, padding: 5 }}>
                          Child
                           <button onClick={() =>value.changeColor('red')} style={{color:'red'}}>红色button>
                           <button onClick={() => value.changeColor('green')} style={{color:'green'}}>绿色button>
                        div>
                    )
                }
            ThemeContext.Consumer>
        )
    }
}

class Father extends Component {
    constructor() {
        super();
        this.state = { color: 'red' };
    }
    changeColor = (color) => {
        this.setState({ color })
    }
    render() {
        let contextVal = {changeColor: this.changeColor,color:this.state.color };
        return (
            <ThemeContext.Provider value={contextVal}>
                <div style={{margin:'10px', border: `5px solid ${this.state.color}`, padding: 5, width: 200 }}>
                    page
                    <Child />
                div>
            ThemeContext.Provider>

        )
    }
}
ReactDOM.render(<Father />, document.querySelector('#root'));
```

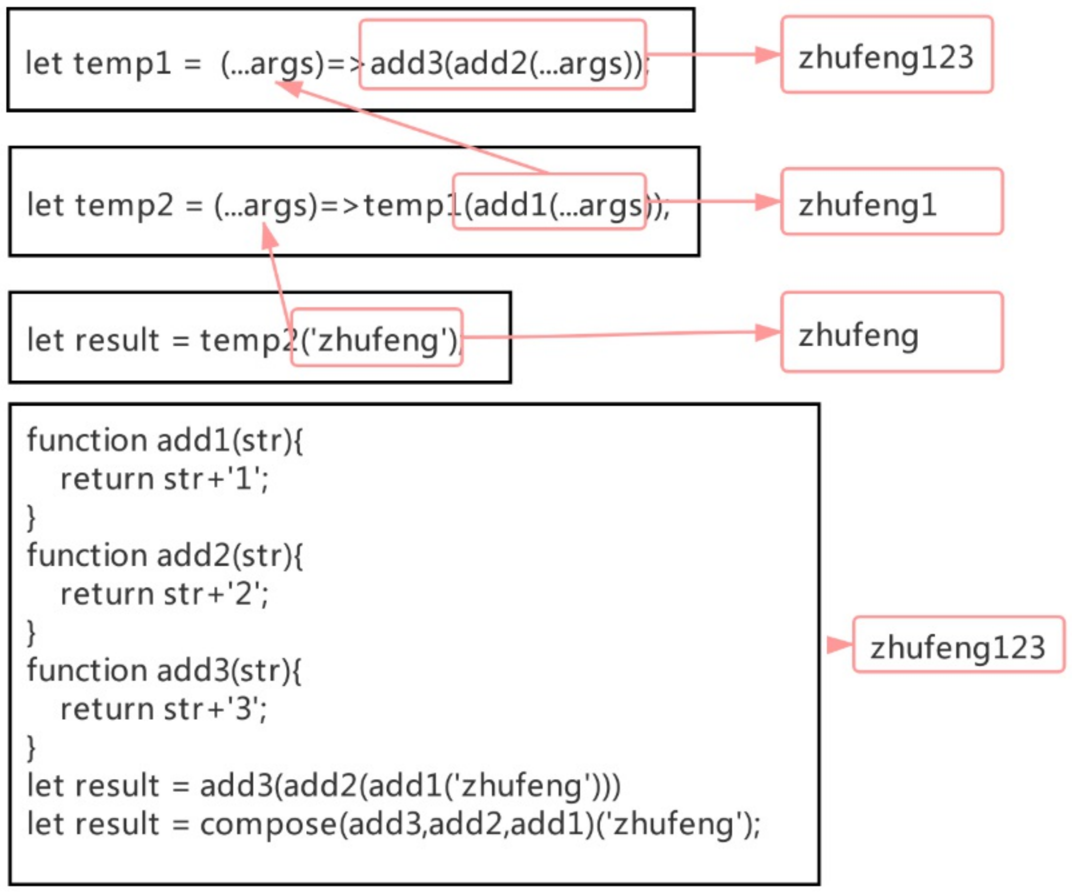
3.3 useReducer

- 接收一个形如 `(state, action) => newState` 的 reducer，并返回当前的 state 以及与其配套的 dispatch 方法

```
const [state, dispatch] = useReducer(reducer, initialArg);
```

```
import React,{useReducer} from 'react';
import ReactDOM from 'react-dom';
const initialState = {number:0};
const INCREMENT = 'INCREMENT';
const DECREMENT = 'DECREMENT';
function reducer(state, action) {
  switch (action.type) {
    case INCREMENT:
      return {number: state.number + 1};
    case DECREMENT:
      return {number: state.number - 1};
    default:
      throw new Error();
  }
}
function Counter(){
    const [state, dispatch] = useReducer(reducer, initialState);
    return (
        <>
          Count: {state.number}
          <button onClick={() => dispatch({type: INCREMENT})}>+button>
          <button onClick={() => dispatch({type: DECREMENT})}>-button>
        </>
    )
}
ReactDOM.render(<Counter />, document.querySelector('#root'));
```

3.4 compose



```
let temp1 =  (...args)=>add3(add2(...args));
let temp2 = (...args)=>temp1(add1(...args));
let result = temp2('zhufeng')
```

```
function add1(str){
   return str+'1';
}
function add2(str){
   return str+'2';
}
function add3(str){
   return str+'3';
}
let result = add3(add2(add1('zhufeng')))
let result = compose(add3,add2,add1)('zhufeng');
```

```
function add1(str){
    return '1'+str;
}
function add2(str){
    return '2'+str;
}
function add3(str){
    return '3'+str;
}

function compose(...funcs){
    return funcs.reduce((a,b)=>(...args)=>a(b(...args)));
}

let result = compose(add3,add2,add1)('zfpx');
console.log(result);
```

4.原版redux #

```
import React,{Component,useReducer} from 'react';
import ReactDOM from 'react-dom';
import {createStore} from 'redux';
import {Provider,connect} from 'react-redux';
const initialState = { number: 0 };
const INCREMENT = "INCREMENT";
const DECREMENT = "DECREMENT";
function reducer(state = initialState, action) {
  switch (action.type) {
    case INCREMENT:
      return { number: state.number + 1 };
    case DECREMENT:
      return { number: state.number - 1 };
    default:
      return state;
  }
}
let store = createStore(reducer);
function Counter(props) {
  return (
    <>
      <p>{props.number}p>
      <button onClick={props.add}>+button>
    </>
  );
}
let mapStateToProps = state => state;
let mapDispatchToProps = dispatch => ({
  add() {
    dispatch({ type: INCREMENT });
  },
  minus() {
    dispatch({ type: DECREMENT });
  }
});
let ConnectedCounter = connect(
  mapStateToProps,
  mapDispatchToProps
)(Counter);

ReactDOM.render(
  <Provider store={store}>
    <ConnectedCounter />
  Provider>,
  document.querySelector("#root")
);
```

5.hooks版redux

5.1 index.js

```
import React,{Component,useReducer} from 'react';
import ReactDOM from 'react-dom';
- import {createStore} from 'redux';
- import {Provider,connect} from 'react-redux';
+ import { createStore } from "./redux";
const initialState = { number: 0 };
const INCREMENT = "INCREMENT";
const DECREMENT = "DECREMENT";
function reducer(state = initialState, action) {
  switch (action.type) {
    case INCREMENT:
      return { number: state.number + 1 };
    case DECREMENT:
      return { number: state.number - 1 };
    default:
      return state;
  }
}
-  let store = createStore(reducer);
+  let { store, Provider, connect } = createStore(reducer, initialState);
function Counter(props) {
  return (
    <>
      {props.number}
      +
    </>
  );
}
let mapStateToProps = state => state;
let mapDispatchToProps = dispatch => ({
  add() {
    dispatch({ type: INCREMENT });
  },
  minus() {
    dispatch({ type: DECREMENT });
  }
});
let ConnectedCounter = connect(
  mapStateToProps,
  mapDispatchToProps
)(Counter);

ReactDOM.render(

  ,
  document.querySelector("#root")
);
```

5.2 index.js

src\redux\index.js

```
import React from "react";
const Context = React.createContext();
export function createStore(reducer, initialState) {
  let store = {};
  const Provider = props => {
    const [state, dispatch] = React.useReducer(reducer, initialState);
    store.getState = () => {
      return state;
    };
    store.dispatch = dispatch;
    return (
      <Context.Provider value={state}>
        {React.cloneElement(props.children)}
      Context.Provider>
    );
  };

  function connect(mapStatetoProps,mapDispatchToProps) {
    return function(Component) {
      let state = initialState;
      let actions ={};
      return props => {
        if (store.getState) state = mapStatetoProps(store.getState());
        actions = mapDispatchToProps(store.dispatch);
        return <Component {...state} {...props} dispatch={store.dispatch} {...actions}/>;
      };
    };
  }
  return { store, connect, Provider };
}
```

6.hooks版中间件

6.1 index.js

```
import React, { Component, useReducer } from "react";
import ReactDOM from "react-dom";
/* import {createStore} from 'redux';
import {Provider,connect} from 'react-redux'; */
import { createStore,applyMiddleware } from "./redux";
const initialState = { number: 0 };
const INCREMENT = "INCREMENT";
const DECREMENT = "DECREMENT";
function reducer(state = initialState, action) {
  switch (action.type) {
    case INCREMENT:
      return { number: state.number + 1 };
    case DECREMENT:
      return { number: state.number - 1 };
    default:
      return state;
  }
}
+ let logger = store=>next=>action=>{
+   console.log(`%c prev state`,`color: #a3a3a3; font-weight: bold`,store.getState());
+   console.log(`%c action`,`color: #7fbedf; font-weight: bold`,store.getState());
+   next(action);
+   console.log(`%c next state`,`color: #9cd69b; font-weight: bold`,store.getState());
+ }
+ let thunk = store=>next=>action=>{
+  if(typeof action === 'function'){
+    return action(store.dispatch,store.getState);
+  }
+  return next(action);
+ }
+ let promise = store=>next=>action=>{
+   if(action.then){
+     return action.then(store.dispatch);
+   }
+   return next(action);
+ }
+ //let { store, Provider, connect } = createStore(reducer, initialState);
+ let { store, Provider, connect } = applyMiddleware(thunk,promise,logger)(createStore)(reducer, initialState);
function Counter(props) {
  return (
    <>
      {props.number}
      +
+       store.dispatch(function(dispatch,getState){
+             setTimeout(() => {
+               dispatch({type:INCREMENT});
+             }, 1000);
+       })}>异步+1
+       store.dispatch(new Promise(function(resolve,reject){
+               setTimeout(() => {
+                 resolve({type:INCREMENT});
+               }, 1000);
+       }))}>Promise+1
+     </>
+   );
}
let mapStateToProps = state => state;
let mapDispatchToProps = dispatch => ({
  add() {
    dispatch({ type: INCREMENT });
  },
  minus() {
    dispatch({ type: DECREMENT });
  }
});
let ConnectedCounter = connect(
  mapStateToProps,
  mapDispatchToProps
)(Counter);

ReactDOM.render(

  ,
  document.querySelector("#root")
);
```

6.2 redux

```
import React from "react";
const Context = React.createContext();
+ function compose(...funcs) {
+   return funcs.reduce((a, b) => (...args) => a(b(...args)));
+ }
+ export function applyMiddleware(...middlewares){
+     return createStore => (...args)=>{
+         const { store, connect, Provider } = createStore(...args);
+         let dispatch;
+         const middlewareAPI= {
+          getState:()=>store.getState(),
+          dispatch:(...args)=>dispatch(...args)
+         }
+         const chain = middlewares.map(middleware=>middleware(middlewareAPI));
+         dispatch = compose(...chain)((...args)=>store._dispatch(...args));
+         store.dispatch = dispatch;
+         return {
+            store, connect, Provider
+         }
+     }
+ }
export function createStore(reducer, initialState) {
  let store = {};
  const Provider = props => {
    const [state, dispatch] = React.useReducer(reducer, initialState);
    store.getState = () => {
      return state;
    };
+    store._dispatch = dispatch;
    return (

        {React.cloneElement(props.children)}

    );
  };

  function connect(mapStatetoProps,mapDispatchToProps) {
    return function(Component) {
      let state = initialState;
      let actions ={};
+       return props => {
+         if (store.getState) state = mapStatetoProps(store.getState());
+         actions = mapDispatchToProps(store.dispatch);
+         return ;
+       };
    };
  }
  return { store, connect, Provider };
}
```