---

link: null
title: 珠峰架构师成长计划
description: src\recoil\index.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=108 sentences=238, words=1603

1.初始化项目

```
create-react-app zhufeng-react-state-examples
cd zhufeng-react-state-examples
cnpm start
```

2.Context

- Context 通过组件树提供了一个传递数据的方法，从而避免了在每一个层级手动的传递 props 属性

2.1 使用

```
import React, { useState, createContext, useContext } from 'react';
import ReactDOM from 'react-dom';
const TodosContext = createContext();

export default function App() {
  const [todoList, setTodoList] = useState([])
  function addTodo(text) {
    setTodoList([...todoList, text])
  }
  return (
    <TodosContext.Provider value={{ todoList, addTodo }}>
      <TodoApp />
    TodosContext.Provider>
  );
}

function TodoApp() {
  const { todoList, addTodo } = useContext(TodosContext);
  const [text, setText] = useState('')
  return (
    <div>
      <button onClick={() => { addTodo(text); setText('') }}>增加button>
      <input value={text} onChange={event => setText(event.target.value)} />
      <ul>
        {
          todoList.map(item => <li key={item}>{item}li>)
        }
      ul>
    div>
  );
}
ReactDOM.render(<App />, document.getElementById('root'));
```

3.Recoil

- [recoil](https://recoiljs.org/docs/introduction/motivation)解决 React全局数据流管理的问题，采用分散管理原子状态的设计模式，支持派生数据。

3.1 安装

```
npm install recoil --save
```

3.2 使用

```
import React, { useState } from 'react';
import { RecoilRoot, atom, useRecoilState } from './recoil';
import ReactDOM from 'react-dom';

const todoListState = atom({
  key: 'todoList',
  default: [],
});

function TodoApp() {

  const [todoList, setTodoList] = useRecoilState(todoListState);
  const [input, setInput] = useState('')
  function addTodo() {
    setTodoList([...todoList, input]);
    setInput('');
  }
  return (
    <div>
      <button onClick={addTodo}>添加button>
      <input value={input} onChange={event => setInput(event.target.value)} />
      <ul>
        {
          todoList.map(item => <li key={item}>{item}li>)
        }
      ul>
    div>
  );
}
ReactDOM.render(
<RecoilRoot>
  <TodoApp />
RecoilRoot>,document.getElementById('root'));
```

3.3 实现

3.3.1 recoil\index.js

src\recoil\index.js

```
import  RecoilRoot from './RecoilRoot';
import  atom from './atom';
import  useRecoilState from './useRecoilState';
export {
    RecoilRoot,
    atom,
    useRecoilState
}
```

3.3.2 RecoilRoot.js

src\recoil\RecoilRoot.js

```
import React,{useRef} from 'react';
import AppContext from './AppContext';

function RecoilRoot({ children }) {
    const state={};
    const store = {getState:()=>state};
    const storeRef = useRef(store);
    return (
        <AppContext.Provider value= {storeRef}>
            {children}
        AppContext.Provider>
    )
}

export default RecoilRoot;
```

3.3.3 atom.js

src\recoil\atom.js

```
const nodes = new Map();

function atom(options) {
  let value = options.default;
  let node =  {
    key:options.key,
    get:()=>{
      return value;
    },
    set:(newValue)=>{
      value = newValue;
    }
  }
  nodes.set(node.key, node);
  return node;
}

function getNode(key) {
  return nodes.get(key);
}

export default atom;

export {
  getNode
}
```

3.3.4 useRecoilState.js

src\recoil\useRecoilState.js

```
import {useState} from 'react';
import {getNode} from './atom';
function useRecoilState(recoilState){
   return [recoilState.get(),useSetRecoilState(recoilState)];
}

function useSetRecoilState(recoilState){
   let [,forceUpdate] = useState(0);
   return newValue=> {
      getNode(recoilState.key).set(newValue);
      forceUpdate(x=>x+1);
  }
}
export default useRecoilState;
```

4.xstate

- XState (https://xstate.js.org/)是一个状态管理的库，用来描述、控制各种状态
- Xstate以不同的状态为切入点，通过状态切换去处理数据
- XState 提供的图形化工具 (https://xstate.js.org/viz/)可以把程序转换为图片

4.1 有限状态机

- XState 整个核心源自于 StateCharts
- 也就是说我们需要定义好整个程序会有哪些状态，和每个状态能转换到哪种状态以及如何转换

4.2 安装

```
cnpm install xstate @xstate/react --save
```

4.3 toggle

4.3.1 使用

```
import { Machine, interpret } from './xstate';
const lightMachine = Machine({
  id: 'toggle',
  initial: 'close',
  states: {
      close: {
          on: {CLICK: 'open'}
      },
      open: {
          on: {CLICK: 'close'}
      }
  }
});

const lightService = interpret(lightMachine).onTransition(state =>
  console.log(state.value)
);

lightService.start();
lightService.send({type:'CLICK'});
lightService.send({type:'CLICK'});
```

4.3.2 实现

4.3.2.1 xstate1\index.js

xstate1\index.js

```
import Machine from './Machine';
import interpret from './interpret';

export {
    Machine,
    interpret,
}
```

4.3.2.2 Machine.js

src\xstate1\Machine.js

```

function Machine(config) {
    return new StateNode(config);
}

class StateNode{
    constructor(config,machine,value){
        this.config = config;
        this.initial = config.initial
        this.value = value||config.initial;
        this.machine = machine||this;
        this.on = config.on;
        let states = {};
        if(config.states){
            for(let key in config.states){
                states[key]=new StateNode(config.states[key],this.machine,key);
            }
        }
        this.states = states;
    }
    next=(event)=>{
        let {type} = event;
        let nextState =  this.on[type];
        return this.getStateNode(nextState);
    }
    getStateNode =  (stateKey) =>{
        return this.machine.states[stateKey];
    }
}
export default  Machine;
```

4.3.2.3 interpret.js

src\xstate1\interpret.js

```
var InterpreterStatus = {
    NotStarted:0,
    Running:1,
    Stopped:2
}

class Interpreter{
    listeners=[]
    constructor(machine){
        this.machine = machine;
        this.listeners = new Set();
        this.status = InterpreterStatus.NotStarted;
        this.state = machine.states[machine.initial];
    }
    send = (event)=>{debugger
        this.state = this.state.next(event);
        this.listeners.forEach(l=>l(this.state));
    }
    onTransition(listener) {
        this.listeners.add(listener);
        return this;
    }
    start() {
        this._status = InterpreterStatus.Running;
        return this;
    }
}
function interpret(machine, options) {
    var interpreter = new Interpreter(machine, options);
    return interpreter;
}
export default  interpret;
```

4.4 todos

4.4.1 使用

```
import { Machine, assign, interpret } from './xstate1';
const todosMachine = Machine({
  id: 'todos',
  initial: 'ready',

  context: {
    todoList: [],
    text: ''
  },
  states: {
    ready: {
      on: {
        "CHANGE": {
          actions: [
            assign({
              text: (_, event) => event.value
            })
          ]
        },
        "ADD_TODO": {
          actions: [
            assign({
              text: "",
              todoList: context => [...context.todoList, context.text]
            })
          ]
        }
      }
    }
  }
})

const todoService = interpret(todosMachine).onTransition(state =>
  console.log(state.context)
);
todoService.start();
todoService.send({ type: 'CHANGE', value: 'eat' })
todoService.send({ type: 'ADD_TODO' });
```

4.4.2 实现

4.4.2.1 xstate\index.js

src\xstate1\index.js

```
import Machine from './Machine';
import interpret from './interpret';
+import assign from './assign';
export {
    Machine,
    interpret,
+   assign
}
```

4.4.2.2 Machine.js

src\xstate1\Machine.js

```
function Machine(config) {
    return new StateNode(config);
}

class StateNode{
    constructor(config,machine,value){
        this.config = config;
        this.initial = config.initial
        this.value = value||config.initial;
        this.machine = machine||this;
+       this.context = config.context||this.machine.context;
        this.on = config.on;
        let states = {};
        if(config.states){
            for(let key in config.states){
                states[key]=new StateNode(config.states[key],this.machine,key);
            }
        }
        this.states = states;
    }
    next=(event)=>{
        let {type} = event;
        let nextState =  this.on[type];
+        if(typeof nextState === 'string'){
+            return this.getStateNode(nextState);
+        }else{
+            let actions = nextState.actions;
+            if(Array.isArray(actions)){
+                let context = this.context;
+                let newContext = {};
+                actions.forEach(action=>{
+                    let assignment = action.assignment;
+                    for(let key in assignment){
+                        if(typeof assignment[key] === 'function'){
+                            newContext[key] = assignment[key](context,event);
+                        }else{
+                            newContext[key] = assignment[key];
+                        }
+                    }
+                });
+                Object.assign(context,newContext);
+            }
+            return this;
        }
    }
    getStateNode =  (stateKey) =>{
        return this.machine.states[stateKey];
    }
}
export default  Machine;
```

4.4.2.3 assign.js

src\xstate\assign.js

```
var assign = function (assignment) {
    return {
      type: 'assign',
      assignment: assignment
    };
};
export default assign;
```

4.5 在React中使用

4.5.1 使用

```
+import React from 'react'
import ReactDOM from 'react-dom';
+import { Machine, assign, interpret,useService } from './xstate';
const todosMachine = Machine({
  id: 'todos',
  initial: 'ready',
  context: {
    todoList: [],
    text: ''
  },
  states: {
    ready: {
      on: {
        "CHANGE": {
          actions: [
            assign({
              text: (_, event) => event.value
            })
          ]
        },
        "ADD_TODO": {
          actions: [
            assign({
              text: "",
              todoList: context => [...context.todoList, context.text]
            })
          ]
        }
      }
    }
  }
})

+const service = interpret(todosMachine).start();
+function TodoApp() {
+  const [state, send] = useService(service)
+  const { context: { text, todoList } } = state
+  return (
+
+       send({ type: 'ADD_TODO' })}>添加
+       send({ type: 'CHANGE', value: e.target.value })} />
+
+        {
+          todoList.map(item => {item})
+        }
+
+
+  );
+}
+ReactDOM.render(, document.getElementById('root'));
```

4.5.2 实现

4.5.2.1 useService.js

src\xstate1\useService.js

```
import {useState} from 'react';
export function useService(service) {
    let [,forceUpdate] = useState(0);
    return [service.state, (event)=>{
        service.send(event);
        forceUpdate(x=>x+1);
    }];
}

export default useService;
```

5.redux

5.1 安装

```
cnpm install @reduxjs-toolkit react-redux --save
```

5.2 使用

```
import React, { useState } from 'react'
import ReactDOM from 'react-dom';
import { Provider, useDispatch, useSelector } from 'react-redux'
import { configureStore, createReducer, combineReducers } from '@reduxjs/toolkit'

function App() {
  const [text, setText] = useState('')
  const todoList = useSelector(state => state.todos)
  const dispatch = useDispatch();

  function addTodo() {
    dispatch({ type: 'ADD_TODO', text })
    setText('')
  }
  return (
    <div>
      <button onClick={addTodo}>增加button>
      <input value={text} onChange={e => setText(e.target.value)} />
      <ul>
        {
          todoList.map(item => <li key={item}>{item}li>)
        }
      ul>
    div>
  );
}

const todosReducer = createReducer([], {
  'ADD_TODO': (state, action) => [...state, action.text]
})

const reducers = combineReducers({ todos: todosReducer })
const store = configureStore({ reducer: reducers })
ReactDOM.render(
  <Provider store={store}>
    <App />
  Provider>, document.getElementById('root'));
```

6.mobx

6.1 安装

```
npm install mobx mobx-react-lite --save
```

6.2 使用

```
import React, { useState } from 'react'
import ReactDOM from 'react-dom';
import { observer } from "mobx-react-lite"
import { makeAutoObservable } from 'mobx'

class TodoStore {
  todoList = []
  addTodo(text) {
    this.todoList = [...this.todoList, text]
  }
  constructor() {
    makeAutoObservable(this)
  }
}

const todoStore = new TodoStore()

const App = observer(() => {
  const [input, setInput] = useState('')
  const { todoList } = todoStore
  function addTodo() {
    todoStore.addTodo(input)
    setInput('')
  }
  return (
    <div>
      <button onClick={addTodo}>添加button>
      <input value={input} onChange={event => setInput(event.target.value)} />
      <ul>
        {
          todoList.map(item => <li key={item}>{item}li>)
        }
      ul>
  div>
  )
})
ReactDOM.render(<App />,document.getElementById('root'));
```