# JSX

No need single quote for the Virtual Dom

Using {} to add JS expressions

What is JS expression(表达式)?

It must have a return value!

* a
* a+b
* fun1(2) if return type is void, it will still return undefined
* arr.map()
* function test() {}

what is Code?

* .if(){}
* .for(){}
* .switch(){case:xxxx}

Define class should be using className

Style = {{key:value, key2:value2}}

Can only have one root tag <div></div>

All the tags must close!

Regarding to the first character

* Lowercase

Will render to html tag if there is no corresponding html for it, cause the error

* Uppercase

Will render as corresponding Component. If there is no Component defined, throw error.

# Components

Collections that includes html, js and css and resources (ie. img, font size etc…)

# Functional Component (suitable for simple components)

<div id=”test”></div>

function Mycomponent() {

console.log(this) //this is undefined here as the strict mode is activated after compilation by babel

return <h2>This is a content of functional componenet</h2>

}

ReactDOM.render(<Mycomponent />, document.getElementById(‘test’))

1. React analyze the Component and find the component
2. If that is functional component , call this function, covert VDOM to TDOM and then render in the page

# Class

* Constructor is optional
* If A extends B and there is a constructor in A , then it must call super(..) inside the constructor of A
* All the method put in the prototype object of class

Class Person = {

constructor(name, age) {

this.name = name

this.age = age

}

// normal method

Speak() {

Console.log(“xxx{name}xxxxx{age}”)

}

//can directly add attributes with default values inside the class

//applied to state also

a = 1;

}

const p1 = new Person (‘Tom’, 20);

console.log(p1) => Person{ name:”Tom”, age:18}

p1.speak()

class Student extends Person{

constructor(name, age, grade) {

super(name, age) // must call super if there is constructor

this.grade = grade

}

speak() {

//method override

Console.log(“xxx{name}xxxxx{age}xxxxx{grade}”)

}

}

# Class Component

class Mycomponent extends React.Component {

render( ){

//same as class, 放在类(M)的原型对象上，供实例使用

return <h1>xxxxxxxxxxxxx</h1>

}

}

ReactDOM.render(<Mycomponent />, document.getElementById(‘test’) )

1. React analyze the Component and find the component
2. If that is class component , new this class and use this instance class to call render method
3. Convert the VDOM returned by render() to TDOM

# State

1. Function no need to put () and should be enclosed by {}

class Weather extends React.Component {

constructor(props){

super(props)

this.state = {isHot:true}

}

changeWeather(){

//ChangeWeather()放在**实例原型对象**上

//由于changeWeather作为onClick的callback function， 不是通过实例调用的而是直接调用， 由于类里面的方法默认开启了局部的严格模式（‘use strict’）， 所以changeweather中的this指向的是undefined instead of windows

console.log(this)

}

render (){

console.log(this)

const {isHot} = this.state

return <h1 onClick={this.changeWeather}>Today is {isHot ? 'Hot' : 'Not Hot'}</h1>

} }

Weather w = new Weather()

w.changeWeather()//这里的this指向的是Weather w 实例

## Solution

//解决this的指向问题

// 对象中新的属性（左）， 把原型对象中的changWeather函数绑定到this（现有的对象实例）并赋值为左边的属性（右）

this.changeWeather = this.changeWeather.bind(this)

## Bind

Function demo() {  
 console.log(this);

}

demo()//output:windows 直接调用

const x = demo.bind({a:1, b:2})

x()//output:windows 直接调用, 这是this 已经改成了新的对象也就是{a:1, b:2}

# setState

State can not change directly, see bad example below

~~const isHot = this.state.isHot~~

~~this.state.isHot = !isHot~~

Have to use pre-set API to change the state => **setState**

//Update operation here is combine/merge not replace, it will update the item and maintain existing items;

this.setState(){isHot:!isHot}

constructor only called once when creating

render called 1 + n times

changeWeather() called n times

# Simplify

class Weather extends React.Component{

    // constructor can be omitted!!!!!!

    // constructor(props) {

    //     super(props)

    //     this.state = {isHot:false, wind:'strong wind'}

    //     this.changeWeather = this.changeWeather.bind(this)

    // }

    state = {isHot:false, wind:'strong wind'}

    render(){

        const {isHot, wind} = this.state;

        return <h1 onClick = {this.changeWeather}>Today is {isHot ? 'Hot' : 'Not Hot'}, {wind}</h1>

    }

    //Self-defined method

    //assign function to variable(this changeWeather is under instance weather instead of class prototype) with arrow function

    //here we dun have **this** keyword in the arrow function, so this will refer to its caller which instance weather

    changeWeather = () => {

        //this refer to caller

        const isHot = this.state.isHot

        this.setState({isHot:!isHot})

    }

}

ReactDOM.render(<Weather/>, document.getElementById('test'))

# Props

**Props is read-only**

class Person extends React.Component{

    render() {

        //传入的收可以批量传入参数，用时React帮我们整合成实例对象上的props属性上

        const{name, sex, age} = this.props

        return (

            <ul>

                <li>Name: {name}</li>

                <li>Alternative Name: {this.props.name}</li> //Alternative way to get data from props

                <li>Sex: {sex}</li>

                <li>Age: {age}</li>

            </ul>

        )

    }

}

//age = {20} -> number datatype

ReactDOM.render(<Person name='Li' sex='male' age='20'/>, document.getElementById('test'))

ReactDOM.render(<Person name='Li' sex='male' age={20}/>, document.getElementById('test'))

## 批量传递与展开运算符…

**批量传递**

//...展开运算符 这里的...跟原生JS的{...p}是不一样的， 因为这里的{}是react&babel给的， 仅适用于标签属性传递

//原生JS{...object}是克隆对象， 如果不加{} 直接...object -> error

ReactDOM.render(<Person {...p}/>, document.getElementById('test'))

**批量传递（原生JS Example）**

let arr1 = [1,3,5,7,9]

let arr2 = [2,4,6,8,10]

console.log(arr1)// [1,3,5,7,9]

console.log(...arr1)// 1 3 5 7 9 展开数组

let arr3 = [...arr1,...arr2]//[1,3,5,7,9,2,4,6,8,10] //连接数组

//函数传参数

function sum(...numbers) {

    //numbers is array in this case

    numbers.reduce((preVal, currVal) =>{

        return preVal + currVal

    })

}

console.log(sum(1, 2, 3, 4)) //10

let person = {name:'Lee', age:18}

//console.log(...person) -> **error, 展开运算符不可以展开对象**

let person2 = {...person} //复制对象with {},地址不同（深复制）

console.log(person2.name) // Lee

//合并， 复制的同时修改属性

//replace the existed Key-value pair

//add or insert new key-value pair

let person3 = {...person, name:'jack', address:'earth'}

console.log(person3)//{name:'jack', age:18, address:'earth}

## Constructor

constructor(props) {

//构造器是否接收props是否传递给super， 取决于是否希望在<**构造器中**>通过this访问props(例如在构造期内通过this.props 给 state赋值)(**构造器外没影响,as react set .props on the instance from outside immediately after calling the constructor**)和为绑定函数绑定实例（this.changeWeather= this.changeWeather.bind(this)）

 //如果不传的话，this.props(within constructor) 就是undefined

//能省略就省略,outside the constructor一样可以拿到props的数据

        super(props)

    }

# Constraints and Default Values

## For the Class Component

class Person extends React.Component{…..

## Inside the class

    //add static to add additional attributes to the person class

    //设置标签限制

    static protoTypes = {

**name**:React.ProtoTypes.string.isRequired,//Name: required and string

**sex**:React.ProtoTypes.string, // constraint sex: string

**age**:React.ProtoTypes.number, // constraint age number

**speak:**React.ProtoTypes.func, // constraint speak function

    }

    //设置默认的标签属性值

    static defaultProps = {

        sex:'NMNF', // deafult value for sex

        age: 18 // default value for age

    }

## OutSide the class

Person.protoTypes = {

    name:React.ProtoTypes.string.isRequired,//Name: required and string

    sex:React.ProtoTypes.string, // constraint sex: string

    age:React.ProtoTypes.number, // constraint age number

    speak:React.ProtoTypes.func, // constraint speak function

}

//默认的标签属性值

Person.defaultProps = {

    sex:'NMNF', // deafult value for sex

    age: 18 // default value for age

}

## Functional Component

**//functional component(No state and No refs, can have props)**

//参数会被整合成一个props对象

function Person(props) {

    const {name, age, sex} = props

    return (

        <ul>

            <li>Name: {name}</li>

            <li>Sex: {sex}</li>

            <li>Age: {age+1}</li>

        </ul>

    )

}

Person.protoTypes = {

    name:React.ProtoTypes.string.isRequired,//Name: required and string

    sex:React.ProtoTypes.string, // constraint sex: string

    age:React.ProtoTypes.number, // constraint age number

    speak:React.ProtoTypes.func, // constraint speak function

}

//默认的标签属性值

Person.defaultProps = {

    sex:'NMNF', // deafult value for sex

    age: 18 // default value for age

}

//If you want to use **this.props** in the constructor,

//you need to pass props to super.

//Otherwise, it doesn't matter because React sets .props on the instance from the outside immediately after calling the constructor.

//So just simply remove constructor() if useless

# Refs

//Ref with String

class Demo extends React.Component {

    showData = () =>{

        const { input1 } = this.refs

        alert(input1.value)

    }

    showData2 = () =>{

        const {input2} = this.refs

        alert(input2.value)

    }

//define ref to define itself and store in the refs attribute in the instance component

    render(){

        return (

            <div>

                <input ref="input1" type="text" placeholder="click to prompt"></input>

                <button onClick={this.showData}>click to show the data on the left side</button>

                <input ref="input2" onBlur={this.showData2} type="text" placeholder="prompt"></input>

            </div>

        )

    }

}

ReactDOM.render(<Demo />, document.getElementById('root'))

//Ref with Callback function

class Demo extends React.Component {

    showData = () =>{

        const { input1 } = this

        alert(input1.value)

    }

    showData2 = () =>{

        const {input2} = this

        alert(input2.value)

    }

//define ref to define itself and store in the refs attribute in the instance component

    render(){

        return (

            <div>

                //ref = {}

                //这里通过箭头函数，传入当前节点（这个InputTag)并把当前的节点赋值给组件实例自身的input1属性上

                <input ref={ currentNode => this.input1 = currentNode } type="text" placeholder="click to prompt"></input>

                <button onClick={this.showData}>click to show the data on the left side</button>

                <input ref={curr => this.input2 = curr} onBlur={this.showData2} type="text" placeholder="prompt"></input>

            </div>

        )

    }

}

ReactDOM.render(<Demo />, document.getElementById('root'))

# LifeCycle of React

# New Version

Diagram

Description automatically generated

生命周期的三个阶段（新）

**1. 初始化阶段:** 由ReactDOM.render()触发---初次渲染

1. constructor()
2. **getDerivedStateFromProps**
3. render()
4. componentDidMount()

**2. 更新阶段:** 由组件内部this.setSate()或父组件重新render触发

1. **getDerivedStateFromProps**
2. shouldComponentUpdate()
3. render()
4. **getSnapshotBeforeUpdate**
5. componentDidUpdate()

**3. 卸载组件:** 由ReactDOM.unmountComponentAtNode()触发

1. componentWillUnmount()

**2.6.5. 重要的勾子**

1. render：初始化渲染或更新渲染调用
2. componentDidMount：开启监听, 发送ajax请求
3. componentWillUnmount：做一些收尾工作, 如: 清理定时器

**2.6.6. 即将废弃的勾子**

1. componentWillMount
2. componentWillReceiveProps
3. componentWillUpdate

现在使用会出现警告，下一个大版本需要加上UNSAFE\_前缀才能使用，以后可能会被彻底废弃，不建议使用。

# Diffing Algorithm

Minimum comparison point is Tag/Node.

针对的是每个标签

The function of key in the virtual dom.

<li key = 0>xxxxxxx</li>

1. Key是虚拟DOM对象的标识
2. 当状态中的数据发生变化时， react会根据新数据生成新的虚拟DOM， 随后React进行新虚拟DOM与旧虚拟DOM的diff比较，规则如下
   1. 旧虚拟DOM与新虚拟DOM相同的key
      1. 若虚拟DOM中内容没变，直接使用之前的真实DOM
      2. 若虚拟DOM中的内容变了， 则生成新的真实DOM， 随后替换第页面之前的真实DOM
   2. 旧虚拟DOM未能找到与新虚拟DOM相同的key
      1. 根据数据创建新的真实DOM， 随后渲染到页面
3. 比较的最小单位是TAG/NODE

Note:虚拟DOM没有.value值，只有真实DOM才有.value值

## Map时用index作为key，可能引发的问题

1. 若对数据进行逆序添加，逆序删除等破坏顺序的操作， 会产生没必要的真实DOM的更新， 效率低
2. 如果结构中包含输入类的DOM， 界面会有问题（错误的DOM更新）
3. 如果不存在逆序添加，逆序删除等破坏顺序的操作， it’s okay to index as your key

## How to select the key

To use unique id or that is only simple data display

# React Scaffolding

SPA-Single Page application

    <meta charset="utf-8" />

    <!-- %PUBLIC\_URL%  代表public文件夹的路径-->

    <link rel="icon" href="%PUBLIC\_URL%/favicon.ico" />

    <!-- 用于开启理想视口， 用于做移动端网页适配-->

    <meta name="viewport" content="width=device-width, initial-scale=1" />

    <!-- 用于配置浏览器页签/地址栏的颜色 紧支持安卓手机浏览器 -->

    <meta name="theme-color" content="#000000" />

    <meta

      name="description"

      content="Web site created using create-react-app"

    />

    <!-- 仅支持苹果手机，网页添加到主屏幕后的图标 -->

    <link rel="apple-touch-icon" href="%PUBLIC\_URL%/logo192.png" />

    <!-- 应用加壳的配置文件-->

    <link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />

    <title>React App</title>

  </head>

  <body>

    <!-- 若浏览器不支持js， 则展示标签中的内容 -->

    <noscript>You need to enable JavaScript to run this app.</noscript>

    <!-- 根容器， 所有组件都往这里放 -->

    <div id="root"></div>

App.js App组件

Index.js 入口文件

Index.html 主页面

# ToDoList

The Way to write the className and style

style={{backgroundColor:mouse ? '#ddd':'white'}}

style={{display:mouse ? 'block':'none'}}

where component to put state?

If data in the state only used by itself, just put 自身

If data in the state are shared and used by some of the components(more than one), put the state in their common parent(状态提升)

//状态在哪里，操作状态的方法就在哪里

  //父改子 - props传参数变量

  //子改父 - props传方法变量

The difference between defaultChecked and checked or defaultValue and value

Default….只在第一次渲染时起作用

The method for changing the state to put in where the state is

# 配置代理

跨域问题的本质是AJAX拒绝了不同端口的response， 所以我们需要代理（proxy）

Diagram

Description automatically generated

## Add proxy in package.json file(can not set multiple proxy)

“proxy”: <http://localhost:8080(代理的转发出口>)， 如果需求的资源本地没有(root is public folder) 就会向代理端口请求)

## setupProxy.js(under src folder and multiple proxy)

const proxy = require('http-proxy-middleware')

module.exports = function(app){

    app.use(

        proxy('/api1',{//所有带有/api1前缀的请求都会经过这个代理配置转发给target

            target:'http://localhost:5000',//请求转发给谁

            changeOrigin:true,//控制服务器收到的请求头中host的值(如果是3000 -> 5000, true -> host = 5000, false(default) -> host = 3000)

            //有些服务器可能会对自己origin以外的host会有所限制，所以最好加上去 = true

            pathRewrite:{'^/api1':''}//重写请求路径把，/api1前缀变成空字符串(去除请求前缀，must)

        }),

        proxy('/api2',{

            target:'http://localhost:5001',

            changeOrigin:true,

            pathRewrite:{'^/api2':''}

        })

    )

}

# 连续解构赋值

    //const {a:{b:{c}}} = obj 拿到a里面的b里面的c的值（最终的变量名为c）

    //const {a:{b:{c：newname}}} = obj 连续解构赋值+重命名（最终的变量名为newname）

    const {keywordElement:{value:keyword}} = this

# 消息订阅与发布

Pubsubjs libarary（兄弟嘴组件的沟通）

先订阅再发布

谁接收消息谁就订阅消息

谁发送消息谁就发布消息

最后在componentWillUnmount()里取消订阅

import PubSub from 'pubsub-js'

// or when using CommonJS

const PubSub = require('pubsub-js');

Basic example

// create a function to subscribe to topics

var mySubscriber = function (msg, data) {

console.log( msg, data );

};

// add the function to the list of subscribers for a particular topic

// we're keeping the returned token, in order to be able to unsubscribe

// from the topic later on

var token = PubSub.subscribe('MY TOPIC', mySubscriber);

// publish a topic asynchronously

PubSub.publish('MY TOPIC', 'hello world!');

// publish a topic synchronously, which is faster in some environments,

// but will get confusing when one topic triggers new topics in the

// same execution chain

// USE WITH CAUTION, HERE BE DRAGONS!!!

PubSub.publishSync('MY TOPIC', 'hello world!');

Cancel specific subscription

// create a function to receive the topic

var mySubscriber = function (msg, data) {

console.log(msg, data);

};

// add the function to the list of subscribers to a particular topic

// we're keeping the returned token, in order to be able to unsubscribe

// from the topic later on

var token = PubSub.subscribe('MY TOPIC', mySubscriber);

// unsubscribe this subscriber from this topic

PubSub.unsubscribe(token);

# Ajax Request

* xhr(open, send, onstatechange)
  + jQuery
  + axios
* fetch（不是基于xhr请求）

try {

      const response = await fetch(`/api1/search/users2?q=${keyword}`)

      const data = await response.json()

      PubSub.publish('update',{isLoading:false,users:data.items})

    } catch(error){

      PubSub.publish('update',{isLoading:false,err:error.message})

    }

# SPA(Single page application)

Text

Description automatically generated with medium confidence

Single page and multiple components

Routing

Key-value pair which key is routing path and value is component/function

Text, email

Description automatically generated

## Routing is based on the history attribute in the BOM

        // let history = History.createBrowserHistory() //方法一，直接使用H5推出的history身上的API

        let history = History.createHashHistory() //方法二，hash值（锚点）

        function push (path) {

            history.push(path) //push into stack

            return false

        }

        function replace (path) {

            history.replace(path) // replace the path at top of stack

        }

        function back() {

            history.goBack()// pop up the path at top and go to that path

        }

        function forword() {

            history.goForward()//push back the popped out path

        }

# React-router-dom

## Basic function

1. knowing the area of navigation and display
2. change <a> to <Link>
3. display area using Route Tag

<Route path=”/xxxx” component={Demo}/>

1. <App> enclosed by <BrowserRouter> or <HashRouter> is a good practice,用于管理整个路由

# 一般组件与路由组件

A picture containing text, receipt

Description automatically generated

# NavLink

NavLink具有高亮效果，点谁谁highlight，可以通过activeClassName='xx' 修改其default效果

                <NavLink activeClassName='demo' className='list-group-item' to="/about">About</NavLink>

                <NavLink activeClassName='demo' className='list-group-item' to="/home">Home</NavLink>

# NavLink的封装

Text

Description automatically generated

<MyNavLink to='/home' title='Home'>{标签体内容}</MyNavLink>

# Switch

<Switch>

                  {/\* 包裹switch之后，只要path匹配上就不会再往后继续进行查找了，效率比较高 \*/}

                  <Route path='/about' component={About} />

                  <Route path='/home' component={Home} />

                  <Route path='/home' component={Test} />

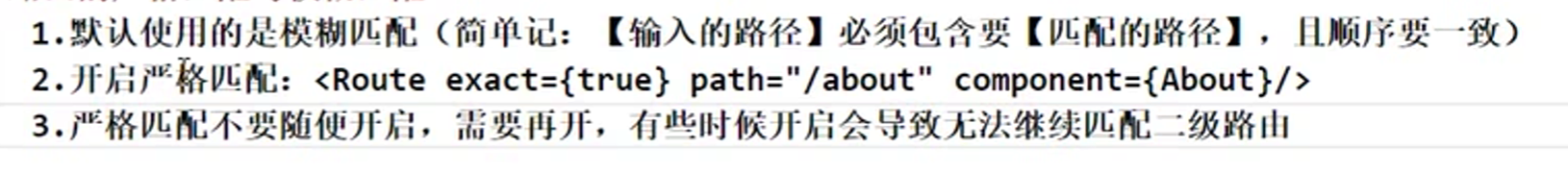
</Switch>



# CSS file loss（解决样式丢失问题）

1. Public/index.html import css file using / instead of ./
2. Public/index.html import css file using %PUBLIC\_URL% instead of ./
3. Using HashRouter

# 严格匹配与模糊匹配（路径）



# Redirect

<Switch>

                  {/\* 包裹switch之后，只要path匹配上就不会再往后继续进行查找了，效率比较高 \*/}

                  <Route path='/about' component={About} />

                  <Route path='/home' component={Home} />

                  {/\* if there is no match, follow the path of redirect \*/}

                  <Redirect to="about"/>

</Switch>

# 嵌套路由/二级路由（搭配模糊匹配）

1. 注册子路由时要写上父路由的path值
2. 路由的匹配是按照注册路由的顺序进行的（start from APP component）

<div>

          <ul className="nav nav-tabs">

            <li>

              <MyNavLink to="/home/news">News</MyNavLink>

            </li>

            <li>

              <MyNavLink to="/home/message">Message</MyNavLink>

            </li>

          </ul>

          {/\* 注册二级路由 \*/}

          <Switch>

            <Route path="/home/news" component={News}/>

            <Route path="/home/message" component={Message}/>

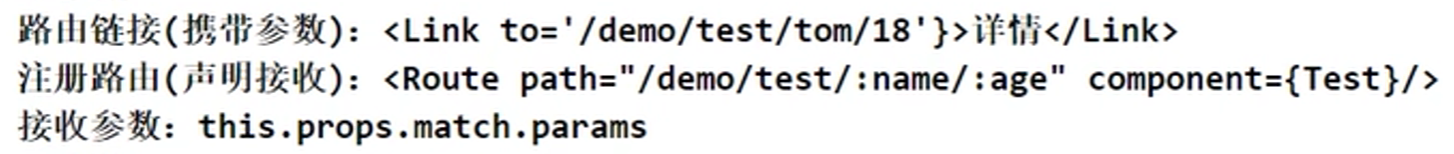
            <Redirect to="/home/news"/>

          </Switch>

</div>

# 路由组件传参

## 1. params参数

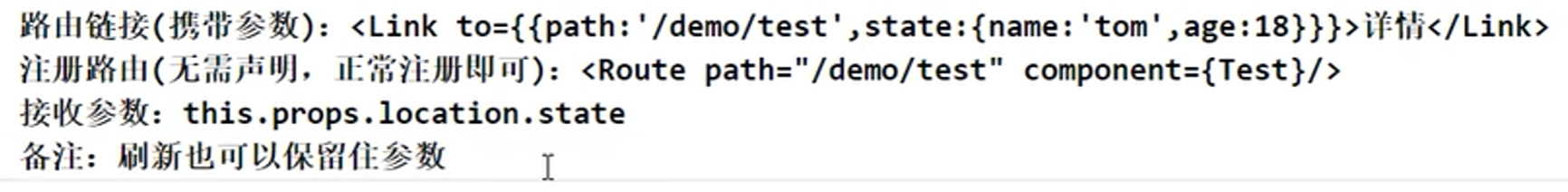


## 2. Search参数

Graphical user interface, text, application

Description automatically generated

## 3. state参数（和组件里的state不一样）



刷新也可以保留住参数的原因是，windows history里面存有记录

# 编程式路由导航

# Text Description automatically generated

this.props.history.go(n)

Go forward when n is positive number and Go back when n is negative number

# WithRouter

//暴露的是withRouther函数调用的返回值

//作用：接受以一般组件，并在一般组件身上加上路由组件所特有的props属性(加工一般组件)

//withRouther的返回值是一个新组件

export default withRouter(Header)

# BrowserRouter vs HashRouter

Text

Description automatically generated

这里的state特指路由里的state -> this.props.location.state 而不是组件内部的state

HashRouter刷新后导致state数据丢失的原因是，其没有用到history的对象

# Ant-design

UI组件库

# Redux

一个用于**状态管理**的JS库，作用为集中式管理react应用中的多个组件的**共享**状态

Text

Description automatically generated

Diagram

Description automatically generated

## Action

* Action object
* Contains 2 attributes
  + Type: 标识属性，值为字符串，唯一必要属性
  + Data: 数据属性，值为任意类型(optional)

## reducer

* 初始化状态，加工状态
* 加工时， 根据prev的state和curr action, 产生新的state的纯函数

## Store

* Combined state, action and reducer
* How to get this object?

Import { createStore } from redux

Import reducer from ‘./reducers’

Const store = createStore(reducer)

* What is the function of the store object?

getState()

dispatch(action): 分发action， 触发reducer调用，产生新的state

subscribe(listener):注册监听， 当产生新的state时，自动调用

**## 1.sum case redux mini**

    (1).remove unnecessary state for the Count component

    (2).create  under folder src

        -redux

            -store.js

            -count\_reducer.js

    (3).store.js

        1).import createStore function from redux and create a store

        2).createStore and pass a reducer as parameter for the service

        3).do not forget to export store object

    (4).count\_reducer.js:

        1).reducer is a function and it has 2 parameters preState and action adn it returns state after the action been proccessed

        2).reducer purpose initilizae the state and processing the state

        3). when the reducer been called for the first time , store is the object who triggered it and preState is undefined and action is @@REDUC/INIT... in this case

    (5).we can detetct the state change for redux in the index.js and it will re-renderd ths pag once it detected the change

        Note:redux only care about the state management and as for we will be responsible for how 状态改变驱动页面的展示

**## 2.sum case redux full**

    newly added files

    1.count\_action.js  用于创建生成action对象

    2.constant.js 放置容易写错的action中的type, 便于管理的同时并减少出错的机率

**## 3.sum case redux async action**

    (1).delay operation pass to action instead Component itself

    (2).when need async action？想要对状态进行操作，但是具体的数据烤async任务返回

    (3).details and steps

        1).npm i redux-thunk and set config in the store object

        2).the created action function now returned another function instead of plain object, the asycn function details will be written in this function

        3).after get the result of async task, dispatch a sync action to execute the data

    (4)Note:异步action不是必须的，完全可以等异步任务有了结果再去分发同步的action

**## 4.sum case basic react-redux**

    (1).UI组件:不能使用任何redux的api， 只负责页面的呈现与交互

        .容器组件:负责与redux通信，将结果交给UI组件

    (2).how to create a container component? using connect function in react-redux

        connect(mapStateToProps,mapDispatchToProps)(UIComponent)

            -mapStateToProps:映射状态，返回值是一个对象

            -mapDispatchToProps:映射操作状态的方法，返回值是个对象

    (3).NOTE1:容器里的store是靠props传进去的，并不是在容器组件中直接引入的

        NOTE2:mapDispatchToProps也可以是一个对象instead of 函数

**## 5.sum case react-redux 优化**

    (1).combined UI Component and Container Component

    (2).no need to pass store object to container component urself

        to use <Procider store={store}><App/></Provider>

    (3).不需要自己监测redux中状态的改变 with react-redux

    (4).mapDispatch can be a object instead of action

    (5).steps

        1).define UI component(不暴露)

        2).import connect to generate container component

            connect(

                state => (){key:value},//映射状态

                {key:xxxxxxAction} //映射操作状态的方法

            )(UI Component)

        3).In the UI Component, use this.props.xxxxx to read/operate the state

**## 6.sum case react-redux 数据共享**

    (1).define Person Component and Count Component and they share the data through redux

    (2).define reducer, action and constant for the Person Component

    (3).key: To combine the Person reducer and Count reducer, use combineReducers

            it is an Object after combination

    (4).reducers are the always the one that are passed to store.

        remeber to take the correct state from redux

**## 7.sum case react-redux dev tools**

    (1).npm i redux-devtools-extension

    (2). config in store object

        import {composeWithDevTools} from 'redux-devtools-extension'

        const store = createStore(allReducer, composeWithDevTools(applyMiddleware(thunk)))

**## 8.sum case react-redux final**

    (1).under folder reducers, can consider to create a index.js file to combine all the exported reducers

        so that in the store.js we only import combined reducers from that index.js file

# React-redux

Diagram

Description automatically generated

NOTE: 这里的状态是指redux里的状态

# 纯函数

Text

Description automatically generated

# 项目打包

Npm build(打包)

Serve build（运行打包后的文件夹build folder）