link: null
title: 珠峰架构师成长计划
description: public\index.html
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=286 sentences=908, words=7089

内容大纲

1. React路由原理

- 不同的路径渲染不同的组件
- 有两种实现方式
    - HashRouter:利用hash实现路由切换
    - BrowserRouter:实现h5 Api实现路由的切换

1.1 HashRouter

- 利用hash实现路由切换

public\index.html

```
Document

    #root{
        border:1px solid red;
    }

    /a
    /b

    window.addEventListener('hashchange',()=>{
        console.log(window.location.hash);
        let pathname = window.location.hash.slice(1);//把最前面的那个#删除
        root.innerHTML = pathname;
    });
```

1.2 BrowserRouter

- 利用h5 Api实现路由的切换

1.2.1 history

- HTML5规范给我们提供了一个history (https://developer.mozilla.org/zh-CN/docs/Web/API/Window/history)接口
- HTML5 History API包括2个方法：`history.pushState()`和`history.replaceState()`，和1个事件`window.onpopstate`

1.2.1.1 pushState

- history.pushState(stateObject, title, url)，包括三个参数
    - 第一个参数用于存储该url对应的状态对象，该对象可在onpopstate事件中获取，也可在history对象中获取
    - 第二个参数是标题，目前浏览器并未实现
    - 第三个参数则是设定的url
- pushState函数向浏览器的历史堆栈压入一个url为设定值的记录，并改变历史堆栈的当前指针至栈顶

1.2.1.2 replaceState

- 该接口与pushState参数相同，含义也相同
- 唯一的区别在于`replaceState`是替换浏览器历史堆栈的当前历史记录为设定的url
- 需要注意的是`replaceState`不会改动浏览器历史堆栈的当前指针

1.2.1.3 onpopstate

- 该事件是window的属性
- 该事件会在调用浏览器的前进、后退以及执行`history.forward`、`history.back`、和`history.go`触发，因为这些操作有一个共性，即修改了历史堆栈的当前指针
- 在不改变document的前提下，一旦当前指针改变则会触发`onpopstate`事件

1.2.1.4 案例

- 浏览器针对每个页面维护一个`History`栈,执行`pushState`函数可压入设定的`url`至栈顶,同时修改当前指针
- 当执行`back`和`forward`操作时，history栈大小并不会改变（history.length不变），仅仅移动当前指针的位置
- 若当前指针在history栈的中间位置(非栈顶)，此时执行pushState会在指针当前的位置添加此条目,并成为新的栈顶

```
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>hash路由title>
head>

<body>
  <div id="root">div>
  <ul>
    <li><a onclick="go('/a')">/aa>li>
    <li><a onclick="go('/b')">/ba>li>
    <li><a onclick="go('/c')">/ca>li>
    <li><a onclick="forward()">前进a>li>
    <li><a onclick="back()">后退a>li>
  ul>
  <script>

    function render() {
      root.innerHTML = window.location.pathname;
    }

    window.onpopstate = render;
    let historyObj = window.history;
    let oldPushState = historyObj.pushState;
    historyObj.pushState = function (state, title, url) {
      oldPushState.apply(history, arguments);
      render();
    }
    function go(path) {
      historyObj.pushState({}, null, path);
    }
    function forward() {
      historyObj.go(1);

    }
    function back(path) {
      historyObj.go(-1);

    }

  script>
body>

html>
```

2.使用基本路由

- https://create-react-app.dev (https://create-react-app.dev)
- https://reactrouter.com/docs/en/v6 (https://reactrouter.com/docs/en/v6)
- https://github.com/remix-run/react-router (https://github.com/remix-run/react-router)

2.1 安装

```
npm i react-router-dom --save
```

2.2 src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { HashRouter, BrowserRouter, Routes, Route } from 'react-router-dom';
import Home from './components/Home';
import User from './components/User';
import Profile from './components/Profile';
ReactDOM.render(
  <HashRouter>
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/user" element={<User />} />
      <Route path="/profile" element={<Profile />} />
    Routes>
  HashRouter>
  , document.getElementById('root'));
```

2.3 Home.js

src\components\Home.js

```
import React from 'react';
function Home(props) {
    console.log(props);
    return (
        <div>Homediv>
    )
}
export default Home;
```

2.4 User.js

src\components\User.js

```
import React from 'react';
function User() {
    return (
        <div>Userdiv>
    )
}
export default User;
```

2.5 Profile.js

src\components\Profile.js

```
import React from 'react';
function Profile() {
    return (
        <div>Profilediv>
    )
}
export default Profile;
```

3.实现基本路由

3.1 react-router-dom\index.js

src\react-router-dom\index.js

```
import React from 'react'
import { Router } from '../react-router';
import { createHashHistory, createBrowserHistory } from "history";
export * from '../react-router';

export function HashRouter({ children }) {
    let historyRef = React.useRef();
    if (historyRef.current == null) {
        historyRef.current = createHashHistory();
    }
    let history = historyRef.current;
    let [state, setState] = React.useState({
        action: history.action,
        location: history.location
    });
    React.useLayoutEffect(() => history.listen(setState), [history]);
    return (
        <Router
            children={children}
            location={state.location}
            navigationType={state.action}
            navigator={history}
        />
    );
}

export function BrowserRouter({ children }) {
    let historyRef = React.useRef();
    if (historyRef.current == null) {
        historyRef.current = createBrowserHistory();
    }
    let history = historyRef.current;
    let [state, setState] = React.useState({
        action: history.action,
        location: history.location
    });
    React.useLayoutEffect(() => history.listen(setState), [history]);
    return (
        <Router
            children={children}
            location={state.location}
            navigationType={state.action}
            navigator={history}
        />
    );
}
```

3.2 src\react-router\index.js

src\react-router\index.js

```
import React from 'react';

const NavigationContext = React.createContext({});

const LocationContext = React.createContext({});

export {
    NavigationContext,
    LocationContext
};
export function Router({ children, location, navigator }) {
  const navigationContext = React.useMemo(() => ({ navigator }), [navigator]);
  const locationContext = React.useMemo(() => ({ location }), [location]);
    return (
    <NavigationContext.Provider value={navigationContext}>
      <LocationContext.Provider value={locationContext} children={children} />
    NavigationContext.Provider>
    );
}
export function Routes({ children }) {
    return useRoutes(createRoutesFromChildren(children));
}
export function useLocation() {
    return React.useContext(LocationContext).location;
}
export function useSearchParams() {
  const location = React.useContext(LocationContext).location;
  const pathname = location.pathname;
  return new URLSearchParams(pathname.split('?')[1]);
}
export function useRoutes(routes) {
    let location = useLocation();
    let pathname = location.pathname || "/";
    for (let i = 0; i < routes.length; i++) {
        let { path, element } = routes[i];
        let match = matchPath(path, pathname);
        if (match) {
            return element;
        }
    }
    return null;
}

export function createRoutesFromChildren(children) {
    let routes = [];
    React.Children.forEach(children, element => {
        let route = {
            path: element.props.path,
            element: element.props.element
        };
        routes.push(route);
    });
    return routes;
}

export function Route(props) {

}
function compilePath(path) {
    let regexpSource = "^" + path;
    regexpSource += "{{content}}quot;;
    let matcher = new RegExp(regexpSource);
    return matcher;
}
export function matchPath(path, pathname) {
    let matcher = compilePath(path);
    let match = pathname.match(matcher);
    if (!match) return null;
    return match;
}
```

4.实现history

4.1 createBrowserHistory.js

src\history\createBrowserHistory.js

```
function createBrowserHistory(){
    const globalHistory = window.history;
    let listeners = [];
    let state;
    function listen(listener){
        listeners.push(listener);
        return ()=>{
            listeners = listeners.filter(item=>item!=listener);
        }
    }
    function go(n){
        globalHistory.go(n);
    }
    window.addEventListener('popstate',()=>{
        let location = {state:globalHistory.state,pathname:window.location.pathname};

        notify({action:"POP",location});
    });
    function goBack(){
        go(-1);
    }
    function goForward(){
        go(1);
    }
    function notify(newState){

        Object.assign(history,newState);
        history.length = globalHistory.length;
         listeners.forEach(listener => listener({ location: history.location }));
    }
    function push(pathname,nextState){
        const action = 'PUSH';
        if(typeof pathname === 'object'){
            state = pathname.state;
            pathname = pathname.pathname;
        }else{
            state=nextState;
        }
        globalHistory.pushState(state,null,pathname);
        let location = {state,pathname};
        notify({action,location});
    }
    const history = {
        action:'POP',
        go,
        goBack,
        goForward,
        push,
        listen,
        location:{pathname:window.location.pathname,state:window.location.state}
    }
    return history;
}
export default createBrowserHistory;
```

4.2 createHashHistory.js

src\history\createHashHistory.js

```
function createHashHistory(){
    let stack = [];
    let index = -1;
    let action = 'POP';
    let state ;
    let listeners = [];
    function listen(listener){
        listeners.push(listener);
        return ()=>{
            listeners = listeners.filter(item=>item!=listener);
        }
    }
    function go(n){
        action = 'POP';
        index+=n;
        let nextLocation = stack[index];
        state= nextLocation.state;
        window.location.hash = nextLocation.pathname;
    }
    let hashChangeHandler = ()=>{
        let pathname = window.location.hash.slice(1);
        Object.assign(history,{action,location:{pathname,state}});
        if(action === 'PUSH'){
            stack[++index]=history.location;
        }
         listeners.forEach(listener => listener({ location: history.location }));
    }
    function push(pathname,nextState){
        action = 'PUSH';
        if(typeof pathname ==='object'){
            state = pathname.state;
            pathname = pathname.pathname
        }else{
            state = nextState;
        }
        window.location.hash = pathname;
    }

    window.addEventListener('hashchange',hashChangeHandler);
    function goBack(){
        go(-1);
    }
    function goForward(){
        go(1);
    }
    const history = {
        action:'POP',
        go,
        goBack,
        goForward,
        push,
        listen,
        location:{},
        location:{pathname:window.location.hash ? window.location.hash.slice(1) : '/',state:undefined}
    }
    if(window.location.hash){
        action = 'PUSH';
        hashChangeHandler();
    }else{
        window.location.hash = '/';
    }
    return history;
}
export default createHashHistory;
```

4.3 history\index.js

src\history\index.js

```
export {default as createBrowserHistory} from './createBrowserHistory';
export {default as createHashHistory} from './createHashHistory';
```

4.4 react-router-dom\index.js

src\react-router-dom\index.js

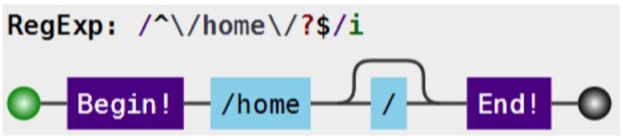
```
import React from 'react'
import { Router } from '../react-router';
+import { createHashHistory, createBrowserHistory } from "../history";
export * from '../react-router';

export function HashRouter({ children }) {
    let historyRef = React.useRef();
    if (historyRef.current == null) {
        historyRef.current = createHashHistory();
    }
    let history = historyRef.current;
    let [state, setState] = React.useState({
        action: history.action,
        location: history.location
    });
    React.useLayoutEffect(() => history.listen(setState), [history]);
    return (

    );
}

export function BrowserRouter({ children }) {
    let historyRef = React.useRef();
    if (historyRef.current == null) {
        historyRef.current = createBrowserHistory();
    }
    let history = historyRef.current;
    let [state, setState] = React.useState({
        action: history.action,
        location: history.location
    });
    React.useLayoutEffect(() => history.listen(setState), [history]);
    return (

    );
}
```

5. path-to-regexp

- regulex (https://jex.im/regulex)
- path-to-regexp (https://www.npmjs.com/package/path-to-regexp)
    - sensitive 是否大小写敏感 (默认值: false)
    - strict 是否允许结尾有一个可选的/ (默认值: false)
    - end 是否匹配整个字符串 (默认值: true)

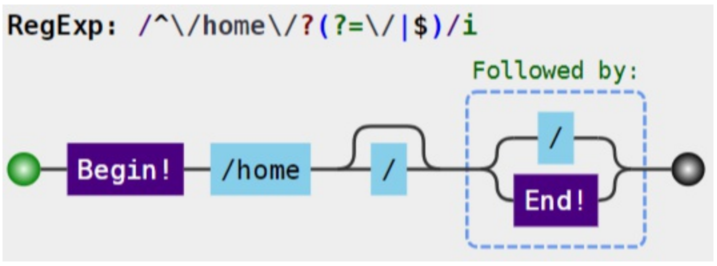
5.1 /home结束

```
let pathToRegExp = require('path-to-regexp');
let regxp = pathToRegExp('/home',[],{end:true});
console.log(regxp);
console.log(regxp.test('/home'));
console.log(regxp.test('/home/2'));
```



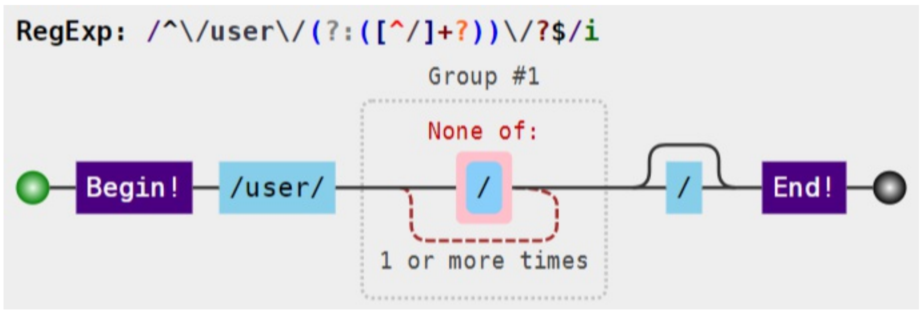
5.2 /home非结束

```
let pathToRegExp = require('path-to-regexp');
let regExp = pathToRegExp('/home',[],{end:false});
console.log(regExp);
console.log(regExp.test('/home'));
console.log(regExp.test('/home/'));
console.log(regExp.test('/home//'));
console.log(regExp.test('/home/2'));
```



5.3 路径参数

```
let params = [];
let regExp = pathToRegExp('/user/:id',params,{end:true});
console.log(regExp,params);
```



5.4 正则匹配

- 是否捕获

表达式 含义 () 表示捕获分组，()会把每个分组里的匹配的值保存起来，使用$n(n是一个数字，表示第n个捕获组的内容) (?:) 表示非捕获分组，和捕获分组唯一的区别在于，非捕获分组匹配的值不会保存起来 (?...) 表示命名捕获分组,反向引用一个命名分组的语法是 \k,在 replace() 方法的替换字符串中反向引用是用 $

```
console.log('1ab'.match(/1[a-z]([b-c])/));

console.log('1ab'.match(/1[a-z](?:[a-z])/));

console.log('11-22'.replace(/(?\d{2})-(?\d{2})/,"$-{{content}}quot;));
```

- 前瞻和后顾

表达式 含义 (?=pattern) 正向肯定查找(前瞻),后面必须跟着什么 (?!pattern) 正向否定查找(前瞻)，后面不能跟着什么 (?

```
console.log('1a'.match(/\d(?=[a-z])[a-z]/));

console.log('1a'.match(/\d(?![A-Z])[a-z]/));

console.log('1a'.match(/(?));

console.log('1a'.match(/(?));

let array = ['1ab'];
array.index = 0;
array.input = '1ab';
array.groups = undefined;
console.log(array);
```

6. 路径参数

6.1 src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { HashRouter, BrowserRouter, Routes, Route } from './react-router-dom';
import Home from './components/Home';
import User from './components/User';
import Profile from './components/Profile';
import Post from './components/Post';
ReactDOM.render(

      } />
      } />
      } />
+     } />

  , document.getElementById('root'));
```

6.2 src\react-router\index.js

src\react-router\index.js

```
import React from 'react';
//导航上下文
const NavigationContext = React.createContext({});
//路径上下文
const LocationContext = React.createContext({});

export {
    NavigationContext,
    LocationContext,
};
export function Router({ children, location, navigator }) {
    let navigationContext = React.useMemo(
        () => ({ navigator }),
        [navigator]
    );
    return (

    );
}
export function Routes({ children }) {
    return useRoutes(createRoutesFromChildren(children));
}
export function useLocation() {
    return React.useContext(LocationContext).location;
}
export function useRoutes(routes) {
    let location = useLocation();//当前的路径对象
    let pathname = location.pathname || "/";//当前的路径
    for (let i = 0; i < routes.length; i++) {
        let { path, element } = routes[i];
        let match = matchPath(path, pathname);
        return match;
    }
    return null;
}

export function createRoutesFromChildren(children) {
    let routes = [];
    React.Children.forEach(children, element => {
        let route = {
            path: element.props.path,
            element: element.props.element
        };
        routes.push(route);
    });
    return routes;
}

export function Route(props) {

}
function compilePath(path) {
+   let paramNames = [];
+   let regexpSource = "^" + path
+       .replace(/:(\w+)/g, (_, key) => {
+           paramNames.push(key);
+           return "([^\\/]+)";
+       });
+    regexpSource += "{{content}}quot;;
    let matcher = new RegExp(regexpSource);
+   return [matcher, paramNames];
}
export function matchPath(path, pathname) {
+   let [matcher, paramNames] = compilePath(path);
    let match = pathname.match(matcher);
    if (!match) return null;
+   let matchedPathname = match[0];
+   let values = match.slice(1);
+   let params = paramNames.reduce(
+       (memo, paramName, index) => {
+           memo[paramName] = values[index];
+           return memo;
+       },
+       {}
+   );
+   return { params, pathname: matchedPathname, path };
}
```

6.3 Post.js

src\components\Post.js

```
import React from 'react';
function Post(props) {
    return (
        <div>Postdiv>
    )
}
export default Post;
```

7.1 src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
+import { HashRouter, BrowserRouter, Routes, Route, Link } from './react-router-dom';
import Home from './components/Home';
import User from './components/User';
import Profile from './components/Profile';
import Post from './components/Post';
ReactDOM.render(

+
+      首页
+      用户管理
+      个人中心
+


      } />
      } />
      } />
      } />

  , document.getElementById('root'));
```

7.2 react-router-dom\index.js

src\react-router-dom\index.js

```
import React from 'react'
+import { Router, useNavigate } from '../react-router';
import { createHashHistory, createBrowserHistory } from "../history";
export * from '../react-router';

export function HashRouter({ children }) {
    let historyRef = React.useRef();
    if (historyRef.current == null) {
        historyRef.current = createHashHistory();
    }
    let history = historyRef.current;
    let [state, setState] = React.useState({
        action: history.action,
        location: history.location
    });
    React.useLayoutEffect(() => history.listen(setState), [history]);
    return (

    );
}

export function BrowserRouter({ children }) {
    let historyRef = React.useRef();
    if (historyRef.current == null) {
        historyRef.current = createBrowserHistory();
    }
    let history = historyRef.current;
    let [state, setState] = React.useState({
        action: history.action,
        location: history.location
    });
    React.useLayoutEffect(() => history.listen(setState), [history]);
    return (

    );
}

+export function Link({ to, children }) {
+  const navigate = useNavigate();//是一个跳转路径的方法
+  return (
+      {
+       event.preventDefault();
+       navigate(to);
+     }} >{children}
+  )
+}
```

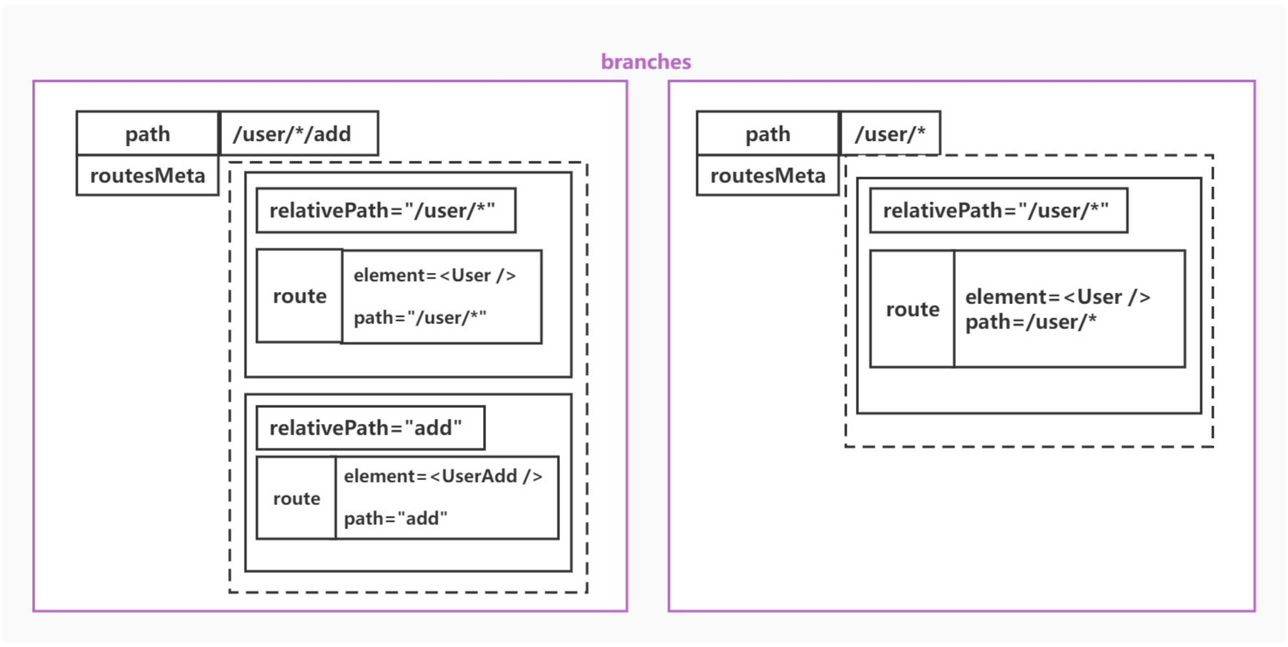
7.3 react-router\index.js

src\react-router\index.js

```
import React from 'react';
//导航上下文
const NavigationContext = React.createContext({});
//路径上下文
const LocationContext = React.createContext({});

export {
    NavigationContext,
    LocationContext
};
export function Router({ children, location, navigator }) {
    let navigationContext = React.useMemo(
        () => ({ navigator }),
        [navigator]
    );
    return (

    );
}
export function Routes({ children }) {
    return useRoutes(createRoutesFromChildren(children));
}
export function useLocation() {
    return React.useContext(LocationContext).location;
}
export function useRoutes(routes) {
    let location = useLocation();//当前的路径对象
    let pathname = location.pathname || "/";//当前的路径
    for (let i = 0; i < routes.length; i++) {
        let { path, element } = routes[i];
        let match = matchPath(path, pathname);
        if (match) {
            return React.cloneElement(element, { ...element.props, match });
        }
    }
    return null;
}

export function createRoutesFromChildren(children) {
    let routes = [];
    React.Children.forEach(children, element => {
        let route = {
            path: element.props.path,
            element: element.props.element
        };
        routes.push(route);
    });
    return routes;
}

export function Route(props) {

}

function compilePath(path) {
    let paramNames = [];
    let regexpSource = "^" + path
        .replace(/:(\w+)/g, (_, key) => {
            paramNames.push(key);
            return "([^\\/]+)";
        });
    regexpSource += "{{content}}quot;;
    let matcher = new RegExp(regexpSource);
    return [matcher, paramNames];
}
export function matchPath(path, pathname) {
    let [matcher, paramNames] = compilePath(path);
    let match = pathname.match(matcher);
    if (!match) return null;
    let matchedPathname = match[0];
    let values = match.slice(1);
    let params = paramNames.reduce(
        (memo, paramName, index) => {
            memo[paramName] = values[index];
            return memo;
        },
        {}
    );
    return { params, pathname: matchedPathname, path };
}

+export function useNavigate() {
+    let { navigator } = React.useContext(NavigationContext);
+    let navigate = React.useCallback((to) => {
+        navigator.push(to);
+    }, [navigator]);
+    return navigate;
+}
```

8. 支持嵌套路由和Outlet

- processon (https://www.processon.com/diagraming/624525f95653bb072bcea329)



8.1 数据结构

8.1.1 routes

```
[
  {
    "path": "/user/*",
    "element": "element",
    "children": [
      {
        "path": "add",
        "element": "element"
      }
    ]
  }
]
```

8.1.2 branches

```
[
{
  "path": "/user/*",
  "routesMeta": [
    {
      "relativePath": "/user/*",
      "route": {
        "path": "/user/*",
        "element": "element",
        "children": "children"
      }
    }
  ]
},
{
  "path": "/user/*/add",
  "routesMeta": [
    {
      "relativePath": "/user/*",
      "route": {
        "path": "/user/*",
        "element": "element",
        "children": "children"
      }
    },
    {
      "relativePath": "add",
      "route": {
        "path": "add",
        "element": "element"
      }
    }
  ]
}
]
```

```
const branches=[
    {path:'/user/*/add',routesMeta:[user*Meta,addMeta]},
    {path:'/user/*/list',routesMeta:[user*Meta,listMeta]},
    {path:'/user/*/detail',routesMeta:[user*Meta,detailMeta]},
    {path:'/user/*',routesMeta:[user*Meta]},
]
```

8.1.3 flattenRoutes

```
function flattenRoutes(routes, branches = []) {
    routes.forEach(route => {
        if (route.children && route.children.length > 0) {
            flattenRoutes(route.children, branches);
        }
        branches.push({ name: route.name });
    });
    return branches;
}
let routes = [
    {
        name: 'A',
        children: [
            {
                name: 'B',
                children: [
                    {
                        name: 'C'
                    }
                ]
            }
        ]
    }
]
console.log(flattenRoutes(routes));
```

8.1.4 remainingPathname

```
let pathname = '/user/add';
let meta = '/user';
let remainingPathname = pathname.slice(meta.length);
```

8.1.5 replace

```
let str = '/user///add';
str = str.replace(/\/\/+/g, '/');
console.log(str);
```

8.1.6 regexpSource

```
let paramNames = [];
let path = '/user';
let end = false;
let regexpSource = "^" + path
    .replace(/\/*\*?$/, '')
    .replace(/^\/*/, '/')
    .replace(/:(\w+)/g, (_, key) => {
        paramNames.push(key);
        return '([^\\/]+)';
    })
if (path.endsWith('*')) {
    paramNames.push('*');
    regexpSource += "(?:\\/(.+)|\\/*){{content}}quot;;
} else {
    regexpSource += end ? "\\/*{{content}}quot; : "(?:\\b|\\/|$)";
}
let matcher = new RegExp(regexpSource);
console.log(matcher);
console.log(paramNames);
let pathname = '/user/add';
let match = pathname.match(matcher);
console.log(match);

let matchedPathname = '/user/add/'
console.log('matchedPathname', matchedPathname);

let pathnameBase = matchedPathname.replace(/(.)\/+$/, '$1');
console.log('pathnameBase', pathnameBase);
```

8.1.7 pathnameBase

- 因为*能匹配所有的内容，所以需要重新计算 pathnameBase

```
let path = '/user/*';
let pathname = '/user/100/detail';
let pathRegexp = /^\/user(?:\/(.+)|\/*)$/;
let match = pathname.match(pathRegexp);
console.log(match);
let matchedPathname = match[0];
let starValue = match[1];
console.log('starValue', starValue);
console.log('matchedPathname', matchedPathname);
console.log('matchedPathname.length', matchedPathname.length);
console.log('value.length', starValue.length);
let pathnameBase = matchedPathname.slice(0, matchedPathname.length - starValue.length)
pathnameBase = matchedPathname.slice(0, 6);
pathnameBase = pathnameBase.replace(/(.)\/+$/, '$1');
console.log('pathnameBase', pathnameBase);
```

8.1.8 computeScore

```
const splatPenalty = -2;
const indexRouteValue = 2;
const paramRegexp = /^:\w+$/;
const dynamicSegmentValue = 3;
const emptySegmentValue = 1;
const staticSegmentValue = 10;
const isSplat = s => s === '*';
function computeScore(path, index) {

    let segments = path.split('/');
    let initialScore = segments.length;
    if (segments.some(isSplat)) {
        initialScore += splatPenalty;
    }
    if (typeof index !== 'undefined') {
        initialScore += indexRouteValue;
    }

    return segments.filter(s => !isSplat(s)).reduce((score, segment) => {
        let currentScope = 0;

        if (paramRegexp.test(segment)) {
            currentScope += dynamicSegmentValue;
        } else {
            if (segment === '') {
                currentScope += emptySegmentValue;
            } else {
                currentScope += staticSegmentValue;
            }
        }
        score += currentScope;
        return score;
    }, initialScore);

}
console.log(computeScore('/user/*', 1));
```

8.1.9 compareIndexes

```
function compareIndexes(a, b) {

    let siblings = a.length === b.length && a.slice(0, -1).every((n, i) => n === b[i]);

    return siblings ? a[a.length - 1] - b[b.length - 1] : 0;
}
console.log(compareIndexes([1, 1, 1], [1, 1, 2]));
```

8.2 src\index.js

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
+import { BrowserRouter, Routes, Route, Link } from './react-router-dom';
import Home from './components/Home';
import User from './components/User';
+import UserAdd from './components/UserAdd';
+import UserList from './components/UserList';
+import UserDetail from './components/UserDetail';
import Profile from './components/Profile';
import Post from './components/Post';
ReactDOM.render(

            首页
            用户管理
            个人中心

            } />
+           } >
+               } />
+               } />
+               } />
+
            } />
            } />

    , document.getElementById('root'));
```

8.3 User.js

src\components\User.js

```
import React from 'react';
+import { Link, Outlet } from '../react-router-dom';
function User() {
    return (
+
+
+               用户列表
+               添加用户
+
+
+
+
+
    )
}
export default User;
```

8.4 src\utils.js

src\utils.js

```
export const UserAPI = {
    list() {
        let usersStr = localStorage.getItem('users');
        let users = usersStr ? JSON.parse(usersStr) : [];
        return users;
    },
    add(user) {
        let users = UserAPI.list();
        users.push(user);
        localStorage.setItem('users', JSON.stringify(users));
    },
    find(id) {
        let users = UserAPI.list();
        return users.find((user) => user.id === id);
    }
}
```

8.5 UserAdd.js

src\components\UserAdd.js

```
import React from 'react'
import { useNavigate } from 'react-router-dom';
import { UserAPI } from '../utils';
export default function UserAdd() {
  const navigate = useNavigate();
  const nameRef = React.useRef();
  const handleSubmit = (event) => {
    event.preventDefault();
    let name = nameRef.current.value;
    UserAPI.add({ id: Date.now() + "", name });
    navigate('/user/list');
  }
  return (
    <form onSubmit={handleSubmit}>
      <input type="text" ref={nameRef} />
      <button type="submit">添加button>
    form>
  )
}
```

8.6 UserDetail.js

src\components\UserDetail.js

```
import React from 'react'
import { useLocation, useParams } from 'react-router-dom';
import { UserAPI } from '../utils';
export default function UserDetail(props) {
  const location = useLocation();
  const { id } = useParams();
  const [user, setUser] = React.useState({});
  React.useEffect(() => {
    let user = location.state;
    if (!user) {
      if (id) {
        user = UserAPI.find(id);
      }
    }
    if (user) setUser(user);
  }, []);
  return (
    <div>
      {user.id}:{user.name}
    div>
  )
}
```

8.7 UserList.js

src\components\UserList.js

```
import React from 'react'
import { Link } from 'react-router-dom';
import { UserAPI } from '../utils';
export default function User() {
  const [users, setUsers] = React.useState([]);
  React.useEffect(() => {
    let users = UserAPI.list();
    setUsers(users);
  }, []);
  return (
    <ul>
      {
        users.map((user, index) => (
          <li key={index}>
            <Link to={`/user/detail/${user.id}`} state={user}>{user.name}Link>
          li>
        ))
      }
    ul>
  )
}
```

8.8 react-router\index.js

src\react-router\index.js

```
import React, { memo } from 'react';
//导航上下文
const NavigationContext = React.createContext();
//路径上下文
const LocationContext = React.createContext();
+//路由上下文
```

```
+const RouteContext = React.createContext();
+export { NavigationContext, LocationContext, RouteContext }
+export function Outlet() {
+  return useOutlet();
+}
+function useOutlet() {
+  let { outlet } = React.useContext(RouteContext);
+  return outlet;
+}
+export function useParams() {
+  let { matches } = React.useContext(RouteContext);
+  let routeMatch = matches[matches.length - 1];
+  return routeMatch ? routeMatch.params : {};
+}
/**
 *
 * @param {*} children 子组件
 * @param {*} location 当前的路径对象
 * @param {*} navigator history对象 go back forward push....

 */
export function Router({ children, location, navigator }) {
  const navigationContext = React.useMemo(() => ({ navigator }), [navigator]);
  const locationContext = React.useMemo(() => ({ location }), [location]);
  return (

  )
}

export function Routes({ children }) {
  const routes = createRoutesFromChildren(children);
  return useRoutes(routes)
}
//
export function Route() { }
export function useLocation() {
  return React.useContext(LocationContext).location;
}
/**
 * 把此路由配置数组渲染成真正的组件
 * @param {*} routes 路由配置数组
 */
export function useRoutes(routes) {
+ //当前的路径对象
+ let location = useLocation();
+ //当前的路径字符串  /user/add
+ let pathname = location.pathname;
+ //用当前的地址栏中的路径和路由进行匹配
+ let matches = matchRoutes(routes, { pathname });
+ //渲染匹配的结果
+ return _renderMatches(matches);
}
+function _renderMatches(matches) {
+  if (!matches) return null;
+  //渲染结果的时候是从右向左执行的
+  //matches=[{route:{element:User}},{route:{element:UserAdd]}}]
+  return matches.reduceRight((outlet, match, index) => {
+    return (
+
+        {match.route.element}
+
+    )
+  }, null);
+}
/**
 * 用当前路径和路由配置进行匹配,获取匹配的结果
 * @param {*} routes 路由配置
 * @param {*} location 当前路径
 */
+function matchRoutes(routes, location) {
+  //获取路径名
+  let pathname = location.pathname;
+  //打平所有的分支路径
+  let branches = flattenRoutes(routes);
+  rankRouteBranches(branches);
+  console.log(branches);
+  //匹配的结果
+  let matches = null;
+  //按分支顺序依次进行匹配，如果匹配上了，直接退出循环，不再进行后续的匹配
+  for (let i = 0; matches == null && i < branches.length; i++) {
+    matches = matchRouteBranch(branches[i], pathname);
+  }
+  return matches;
+}
+function rankRouteBranches(branches) {
+  branches.sort((a, b) => {
+    //如果分数不一样，按分数倒序排列
+    //如果分数一样，只能比过索引
+    return a.score !== b.score ? b.score - a.score : compareIndexes(
+      a.routesMeta.map(meta => meta.childrenIndex),
+      b.routesMeta.map(meta => meta.childrenIndex)
+    );
+  });
+}
+/**
+ * /user/add   routesMeta=[userMeta,addMeta]=>[2,0]
+ * /user/list  routesMeta = [userMeta,listMeta]=>[2,1];
+ */
+function compareIndexes(a, b) {
+  //如果级别数量相等，并且父亲都 一样，说是他们是兄弟
+  let sibling = a.length === b.length && a.slice(0, -1).every((n, i) => n === b[i])
+  //如果是兄弟的话，那和比索引，索引越小级别越高，索引越大，级别越低
+  //如果不是兄弟，那就认为相等的
+  return sibling ? a[a.length - 1] - b[b.length - 1] : 0;
```

```
+}
+/**
+ * 用分支的路径匹配地址栏的路径名
+ * @param {*} branch
+ * @param {*} pathname 完整路径
+ */
+function matchRouteBranch(branch, pathname) {
+  let { routesMeta } = branch;
+  //此分支路径参数对象  path =/:a/:b/:c  pathname=/vA/vB/vC
+  let matchesParams = {};//{a:vA,b:vB,c:vC}
+  let matchedPathname = "/";
+  let matches = [];
+  for (let i = 0; i < routesMeta.length; i++) {
+    //获取当前的meta
+    let meta = routesMeta[i];
+    //判断是否是最后一个meta
+    let end = i === routesMeta.length - 1;
+    //获取剩下的的将要匹配的路径
+    let remainingPathname = matchedPathname === "/" ? pathname : pathname.slice(matchedPathname.length);
+    let match = matchPath({ path: meta.relativePath, end }, remainingPathname);
+    //如果没有匹配上，那就表示匹配失败了
+    if (!match) {
+      return null;
+    }
+    Object.assign(matchesParams, match.params);
+    let route = meta.route;
+    matches.push({
+      params: matchesParams,
+      pathname: joinPaths([matchedPathname, match.pathname]),
+      pathnameBase: joinPaths([matchedPathname, match.pathnameBase]),
+      route
+    });
+    if (match.pathnameBase !== '/') {
+      matchedPathname = joinPaths([matchedPathname, match.pathnameBase]);
+    }
+  }
+  return matches;
+}
+/**
+ * 匹配路径
+ * @param {*} path 路由的路径
+ * @param {*} pathname 当前地址栏中的路径
+ */
+export function matchPath({ path, end }, pathname) {
+  //把路径编译 成正则
+  let [matcher, paramNames] = compilePath(path, end);
+  //匹配结果
+  let match = pathname.match(matcher);
+  //如果没有匹配上结束
+  if (!match) {
+    return null;
+  }
+  //获取匹配的路径
+  let matchedPathname = match[0]; //  /user//
+  //base就是基本路径 /user/  => /user  把结束的一个/或多个/去掉
+  let pathnameBase = matchedPathname.replace(/(.)\/+$/, '$1');
+  //拼出paramNames
+  let values = match.slice(1);
+  let captureGroups = match.slice(1);
+  let params = paramNames.reduce((memo, paramName, index) => {
+    //  /user/*
+    if (paramName === '*') {
+      let splatValue = captureGroups[index] || '';//后面的内容  pathname=/user/add
+      //pathnameBase=matchedPathname=/user/add
+      //重写pathnameBase == /user/add  slice=/user/ /user  截取*之前的串作为后续匹配的父串
+      pathnameBase = matchedPathname.slice(0, matchedPathname.length - splatValue.length).replace(/(.)\/+/, +'$1');
+    }
+    memo[paramName] = values[index];
+    return memo;
+  }, {});
+  return {
+    params,
+    pathname: matchedPathname,//user/add
+    pathnameBase // /user
+  }
+}
function compilePath(path, end) {
  //路径参数的参数名数组 /post/:id paramNames=["id"]
  let paramNames = [];
  let regexpSource = '^' + path
+   .replace(/\/*\*?$/, '') //的 /*或者//* 或者 * 全部转为空  /user/* /user* /user//* /user 在转正则的时候是等价的
+   .replace(/^\/*/, '/')//把开始多个/或者说没有/转成一个/   /user 不变 //user 变/user  user /user
    .replace(
      /:(\w+)/g, (_, key) => {
        paramNames.push(key);
        return "([^\\/]+?)";
      }
    )
+ if (path.endsWith('*')) {
+   paramNames.push('*');
+   regexpSource += path === "*" || path === "/*" ? "(.*){{content}}quot;
+     : "(?:\\/(.+)|\\/*){{content}}quot;;
+   //regexpSource += "(?:\\/(.+)|\\/*){{content}}quot;;
+ } else {
+   regexpSource += end ? "\\/*{{content}}quot; : "(?:\b|\\/|$)";
+ }
  let matcher = new RegExp(regexpSource);
  return [matcher, paramNames];
}
+const isSplat = s => s === '*';
+const splatPenalty = -2;
+const indexRouteValue = 2;
+const paramRe = /^:\w+$/;
+const dynamicSegmentValue = 3;
```

```
+const emptySegmentValue = 1;
+const staticSegmentValue = 10;
+function computeScore(path, index) {
+  let segments = path.split('/'); // /user/add   => ['user','add']
+  let initialScore = segments.length;//分片的长度就是基础分数
+  if (segments.some(isSplat)) {//  /user/* 有星说是通配，分数分降低
+    initialScore += splatPenalty;
+  }
+  if (index) {
+    initialScore += indexRouteValue;
+  }
+ return segments.filter(s => !isSplat(s)).reduce((score, segment) => {
+   return score + (paramRe.test(segment) ? dynamicSegmentValue : segment === '' ? emptySegmentValue : staticSegmentValue);
+ }, initialScore);
}
+/**
+ * 打平所有的分支
+ * @param {*} routes 路由配置
+ */
+function flattenRoutes(routes, branches = [], parentsMeta = [], parentPath = "") {
+  routes.forEach((route, index) => {
+    //定义一个路由元数据
+    let meta = {
+      relativePath: route.path || "",//路径相对父路径的路径 UserAdd relativePath=add
+      route, //路由对象
+      childrenIndex: index,
+    }
+    //现在我们的routes其实只有一个元素,/user/*  parentPath=''  relativePath=/user/*
+    //path=/user/*
+    //把父路径加上自己的相对路径构建成匹配的完整路径
+    let path = joinPaths([parentPath, meta.relativePath]);
+    //在父meta数组中添加自己这个meta
+    let routesMeta = parentsMeta.concat(meta);
+    //如果有子路由的话，递归添加到 branches分支数组中
+    if (route.children && route.children.length > 0) {
+      flattenRoutes(route.children, branches, routesMeta, path);
+    }
+    branches.push({
+      path,
+      routesMeta,
+      score: computeScore(path, route.index)
+    });
+  });
+  return branches;
+}
+function joinPaths(paths) {
+  // ['/user/*/','/add']=> /user/*/add
+  return paths.join('/').replace(/\/\/+/g, '/');
+}

export function createRoutesFromChildren(children) {
  let routes = [];
  React.Children.forEach(children, (element) => {
    let route = {
      path: element.props.path,//       /user 此路由对应的路径
      element: element.props.element //  此路由对应的元素
    }
+    if (element.props.children) {
+      route.children = createRoutesFromChildren(element.props.children);
+    }
    routes.push(route);
  });
  return routes;
}
export function useNavigate() {
  const { navigator } = React.useContext(NavigationContext);
  const navigate = React.useCallback((to) => navigator.push(to), [navigator]);
  return navigate;
}
```

9.1 public\index.html

public\index.html

```

  React App
+ </span>
<span class="hljs-addition">+   .active {</span>
<span class="hljs-addition">+      color: red;</span>
<span class="hljs-addition">+   }</span>
<span class="hljs-addition">+
```

9.2 src\index.js

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
+import { BrowserRouter, Routes, Route, NavLink } from './react-router-dom';
import Home from './components/Home';
import User from './components/User';
import UserAdd from './components/UserAdd';
import UserList from './components/UserList';
import UserDetail from './components/UserDetail';
import Profile from './components/Profile';
import Post from './components/Post';
+const activeStyle = { backgroundColor: 'green' };
+const activeClassName = 'active';
+const activeNavProps = {
+  style: ({ isActive }) => isActive ? activeStyle : {},
+  className: ({ isActive }) => isActive ? activeClassName : ''
+}
ReactDOM.render(

+          首页
+          用户管理
+          个人中心

            } />
            } >
                } />
                } />
                } />

            } />
            } />

    , document.getElementById('root'));
```

9.3 src\react-router-dom\index.js

src\react-router-dom\index.js

```
import React from 'react'
+import { Router, useNavigate, useLocation } from '../react-router';
import { createHashHistory, createBrowserHistory } from "../history";
export * from '../react-router';

export function HashRouter({ children }) {
    let historyRef = React.useRef();
    if (historyRef.current == null) {
        historyRef.current = createHashHistory();
    }
    let history = historyRef.current;
    let [state, setState] = React.useState({
        action: history.action,
        location: history.location
    });
    React.useLayoutEffect(() => history.listen(setState), [history]);
    return (

    );
}

export function BrowserRouter({ children }) {
    let historyRef = React.useRef();
    if (historyRef.current == null) {
        historyRef.current = createBrowserHistory();
    }
    let history = historyRef.current;
    let [state, setState] = React.useState({
        action: history.action,
        location: history.location
    });
    React.useLayoutEffect(() => history.listen(setState), [history]);
    return (

    );
}
export function Link({ to, ...rest }) {
    let navigate = useNavigate();
    function handleClick() {
        navigate(to);
    }
    return (

    );
}
+/**
+ *
+ * @param {*} className 类名 可以是固定的字符串，也可以是一个函数，函数的参数是isActive
+ * @param {*} end 是否结束
+ * @param {*} style 行内样式 可以是固定的字符串，也可以是一个函数，函数的参数是isActive
+ * @param {*} to 点击导航跳转的路径
+ * @param {*} children 子组件
+ */
+export function NavLink({ className: classNameProp = '', end = false, style: styleProp = {}, to, children, ...+rest }) {
+  let location = useLocation();
+  let path = { pathname: to };
+  let locationPathname = location.pathname;//当前的路径
+  let toPathname = path.pathname;//当前导航想要跳转的路径
+  //如果路径一样，或者 不结束，并且当前的路径是以to开头的，并且下一个字符/，也就是路径路径分隔符
+  let isActive = locationPathname === toPathname
+    || (!end && locationPathname.startsWith(toPathname) && locationPathname.charAt(toPathname.length) === '/')
+  let className;
+  if (typeof classNameProp === 'function') {
+    className = classNameProp({
+      isActive
+    });
+  }
+  let style;
+  if (typeof styleProp === 'function') {
+    style = styleProp({
+      isActive
+    });
+  }
+  return (
+    {children}
+  )
+}
```

10. 跳转和重定向

10.1 src\index.js

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
+import { BrowserRouter, Routes, Route, NavLink, Navigate } from './react-router-dom';
import Home from './components/Home';
import User from './components/User';
import UserAdd from './components/UserAdd';
import UserList from './components/UserList';
import UserDetail from './components/UserDetail';
import Profile from './components/Profile';
import Post from './components/Post';
const activeStyle = { backgroundColor: 'green' };
const activeClassName = 'active';
const activeNavProps = {
  style: ({ isActive }) => isActive ? activeStyle : {},
  className: ({ isActive }) => isActive ? activeClassName : ''
}
ReactDOM.render(

          首页
          用户管理
          个人中心

            } />
            } >
                } />
                } />
                } />

            } />
            } />
+           } />

    , document.getElementById('root'));
```

10.2 Home.js

src\components\Home.js

```
import React from 'react';
+import { useNavigate } from '../react-router-dom';
function Home(props) {
+   let navigate = useNavigate();
+   function navigateTo() {
+       navigate('/profile');
+   };
    return (

            Home
+           跳转到/profile

    )
}
export default Home;
```

10.3 react-router\index.js

src\react-router\index.js

```
import React, { memo } from 'react';
//导航上下文
const NavigationContext = React.createContext();
//路径上下文
const LocationContext = React.createContext();
//路由上下文
const RouteContext = React.createContext();
export { NavigationContext, LocationContext, RouteContext }
export function Outlet() {
  return useOutlet();
}
function useOutlet() {
  let { outlet } = React.useContext(RouteContext);
  return outlet;
}
export function useParams() {
  let { matches } = React.useContext(RouteContext);
  let routeMatch = matches[matches.length - 1];
  return routeMatch ? routeMatch.params : {};
}
/**
 *
 * @param {*} children 子组件
 * @param {*} location 当前的路径对象
 * @param {*} navigator history对象 go back forward push....

 */
export function Router({ children, location, navigator }) {
  const navigationContext = React.useMemo(() => ({ navigator }), [navigator]);
  const locationContext = React.useMemo(() => ({ location }), [location]);
  return (

  )
}

export function Routes({ children }) {
  const routes = createRoutesFromChildren(children);
  return useRoutes(routes)
}
//
export function Route() { }
export function useLocation() {
  return React.useContext(LocationContext).location;
}
/**
 * 把此路由配置数组渲染成真正的组件
 * @param {*} routes 路由配置数组
 */
export function useRoutes(routes) {
  //当前的路径对象
  let location = useLocation();
  //当前的路径字符串  /user/add
  let pathname = location.pathname;
  //用当前的地址栏中的路径和路由进行匹配
  let matches = matchRoutes(routes, { pathname });
  //渲染匹配的结果
  return _renderMatches(matches);
}
function _renderMatches(matches) {
  if (!matches) return null;
  //渲染结果的时候是从右向左执行的
  //matches=[{route:{element:User}},{route:{element:UserAdd]}}]
  return matches.reduceRight((outlet, match, index) => {
    return (

        {match.route.element}

    )
  }, null);
}
/**
 * 用当前路径和路由配置进行匹配,获取匹配的结果
 * @param {*} routes 路由配置
 * @param {*} location 当前路径
 */
function matchRoutes(routes, location) {
  //获取路径名
  let pathname = location.pathname;
  //打平所有的分支路径
  let branches = flattenRoutes(routes);
  rankRouteBranches(branches);
  console.log(branches);
  //匹配的结果
  let matches = null;
  //按分支顺序依次进行匹配，如果匹配上了，直接退出循环，不再进行后续的匹配
  for (let i = 0; matches == null && i < branches.length; i++) {
    matches = matchRouteBranch(branches[i], pathname);
  }
  return matches;
}
function rankRouteBranches(branches) {
  branches.sort((a, b) => {
    //如果分数不一样，按分数倒序排列
    //如果分数一样，只能比过索引
    return a.score !== b.score ? b.score - a.score : compareIndexes(
      a.routesMeta.map(meta => meta.childrenIndex),
      b.routesMeta.map(meta => meta.childrenIndex)
    );
  });
}
/**
 * /user/add   routesMeta=[userMeta,addMeta]=>[2,0]
 * /user/list  routesMeta = [userMeta,listMeta]=>[2,1];
 */
function compareIndexes(a, b) {
  //如果级别数量相等，并且父亲都 一样，说是他们是兄弟
  let sibling = a.length
  //如果是兄弟的话，那和比索引，索引越小级别越高，索引越大，级别越低
  //如果不是兄弟，那就认为相等的
  return sibling ? a[a.length - 1] - b[b.length - 1] : 0;
}
/**
 * 用分支的路径匹配地址栏的路径名
 * @param {*} branch
 * @param {*} pathname 完整路径
 */
function matchRouteBranch(branch, pathname) {
  let { routesMeta } = branch;
  //此分支路径参数对象  path =/:a/:b/:c  pathname=/vA/vB/vC
  let matchesParams = {};//{a:vA,b:vB,c:vC}
  let matchedPathname = "/";
  let matches = [];
  for (let i = 0; i < routesMeta.length; i++) {
    //获取当前的meta
    let meta = routesMeta[i];
    //判断是否是最后一个meta
    let end = i
    //获取剩下的的将要匹配的路径
    let remainingPathname = matchedPathname
    let match = matchPath({ path: meta.relativePath, end }, remainingPathname);
    //如果没有匹配上，那就表示匹配失败了
    if (!match) {
      return null;
    }
    Object.assign(matchesParams, match.params);
    let route = meta.route;
    matches.push({
      params: matchesParams,
      pathname: joinPaths([matchedPathname, match.pathname]),
      pathnameBase: joinPaths([matchedPathname, match.pathnameBase]),
      route
    });
    if (match.pathnameBase !== '/') {
      matchedPathname = joinPaths([matchedPathname, match.pathnameBase]);
    }
  }
  return matches;
}
/**
 * 匹配路径
 * @param {*} path 路由的路径
 * @param {*} pathname 当前地址栏中的路径
 */
export function matchPath({ path, end }, pathname) {
  //把路径编译 成正则
  let [matcher, paramNames] = compilePath(path, end);
  //匹配结果
  let match = pathname.match(matcher);
  //如果没有匹配上结束
  if (!match) {
    return null;
  }
  //获取匹配的路径
  let matchedPathname = match[0]; //  /user//
  //base就是基本路径 /user/  => /user  把结束的一个/或多个/去掉
  let pathnameBase = matchedPathname.replace(/(.)\/+$/, '$1');
  //拼出paramNames
  let values = match.slice(1);
  let captureGroups = match.slice(1);
  let params = paramNames.reduce((memo, paramName, index) => {
    //  /user/*
    if (paramName
      let splatValue = captureGroups[index] || '';//后面的内容  pathname=/user/add
      //pathnameBase=matchedPathname=/user/add
      //重写pathnameBase == /user/add  slice=/user/ /user  截取*之前的串作为后续匹配的父串
      pathnameBase = matchedPathname.slice(0, matchedPathname.length - splatValue.length).replace(/(.)\/+/, '$1');
    }
    memo[paramName] = values[index];
    return memo;
  }, {});
  return {
    params,
    pathname: matchedPathname,//user/add
    pathnameBase // /user
  }

}
function compilePath(path, end) {
  //路径参数的参数名数组 /post/:id paramNames=["id"]
  let paramNames = [];
  let regexpSource = '^' + path
    .replace(/\/*\*?$/, '') //的 /*或者//* 或者 * 全部转为空  /user/* /user* /user//* /user 在转正则的时候是等价的
    .replace(/^\/*/, '/')//把开始多个/或者说没有/转成一个/   /user 不变 //user 变/user  user /user
    .replace(
      /:(\w+)/g, (_, key) => {
        paramNames.push(key);
        return "([^\\/]+?)";
      }
    )
  if (path.endsWith('*')) {
    paramNames.push('*');
    // Already matched the initial /, just match the rest
    regexpSource += path
      : "(?:\\/(.+)|\\/*){{content}}quot;; // Don't include the / in params["*"]
    //regexpSource += "(?:\\/(.+)|\\/*){{content}}quot;;
  } else {
    regexpSource += end ? "\\/*{{content}}quot; : "(?:\b|\\/|$)";
  }
  let matcher = new RegExp(regexpSource);
  return [matcher, paramNames];
}
const isSplat = s => s
const splatPenalty = -2;
const indexRouteValue = 2;
const paramRe = /^:\w+$/;
const dynamicSegmentValue = 3;
const emptySegmentValue = 1;
const staticSegmentValue = 10;
function computeScore(path, index) {
  let segments = path.split('/'); // /user/add   => ['user','add']
  let initialScore = segments.length;//分片的长度就是基础分数
  if (segments.some(isSplat)) {//  /user/* 有星说是通配，分数分降低
    initialScore += splatPenalty;
  }
  if (index) {
    initialScore += indexRouteValue;
  }
  //1.过滤*
  //
  return segments.filter(s => !isSplat(s)).reduce((score, segment) => {
    return score + (paramRe.test(segment) ? dynamicSegmentValue : segment
  }, initialScore);
}
/**
 * 打平所有的分支
 * @param {*} routes 路由配置
 */
function flattenRoutes(routes, branches = [], parentsMeta = [], parentPath = "") {
  routes.forEach((route, index) => {
    //定义一个路由元数据
    let meta = {
      relativePath: route.path || "",//路径相对父路径的路径 UserAdd relativePath=add
      route, //路由对象
      children
    }
    //现在我们的routes其实只有一个元素,/user/*  parentPath=''  relativePath=/user/*
    //path=/user/*
    //把父路径加上自己的相对路径构建成匹配的完整路径
    let path = joinPaths([parentPath, meta.relativePath]);
    //在父meta数组中添加自己这个meta
    let routesMeta = parentsMeta.concat(meta);
    //如果有子路由的话，递归添加到 branches分支数组中
    if (route.children && route.children.length > 0) {
      flattenRoutes(route.children, branches, routesMeta, path);
    }
    branches.push({
      path,
      routesMeta,
      score: computeScore(path, route.index)
    });
  });
  return branches;
}
function joinPaths(paths) {
  // ['/user/*/','/add']=> /user/*/add
  return paths.join('/').replace(/\/\/+/g, '/');
}

export function createRoutesFromChildren(children) {
  let routes = [];
  React.Children.forEach(children, (element) => {
    let route = {
      path: element.props.path,//       /user 此路由对应的路径
      element: element.props.element //  此路由对应的元素
    }
    if (element.props.children) {
      route.children = createRoutesFromChildren(element.props.children);
    }
    routes.push(route);
  });
  return routes;
}
export function useNavigate() {
  // navigator history
  // Navigate动词表示导航 或者叫跳转
  const { navigator } = React.useContext(NavigationContext);
  const navigate = React.useCallback((to) => navigator.push(to), [navigator]);
  return navigate;
}
+export function Navigate({ to }) {
+  let navigate = useNavigate();
+  React.useLayoutEffect(() => {
+    navigate(to)
+  });
+  return null;
+}
```

11. 受保护路由

11.1 src\index.js

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter, Routes, Route, NavLink, Navigate } from './react-router-dom';
import Home from './components/Home';
import User from './components/User';
import UserAdd from './components/UserAdd';
import UserList from './components/UserList';
import UserDetail from './components/UserDetail';
import Profile from './components/Profile';
import Post from './components/Post';
+import Protected from './components/Protected';
+import Login from './components/Login';
const activeStyle = { backgroundColor: 'green' };
const activeClassName = 'active';
const activeNavProps = {
  style: ({ isActive }) => isActive ? activeStyle : {},
  className: ({ isActive }) => isActive ? activeClassName : ''
}
ReactDOM.render(

          首页
          用户管理
          个人中心

            } />
            } >
                } />
                } />
                } />

            } />
+           } />
+           } />
            } />

    , document.getElementById('root'));
```

11.2 Login.js

src\components\Login.js

```
import React from 'react';
import { useNavigate, useLocation } from '../react-router-dom';
function Login() {
    let navigation = useNavigate();
    let location = useLocation();
    const login = () => {
        localStorage.setItem('login', 'true');
        let to = '/';
        if (location.state) {
            to = location.state.from || '/';
        }
        navigation(to);
    }
    return (
        <button onClick={login}>登录button>
    )
}
export default Login;
```

11.3 Protected.js

src\components\Protected.js

```
import React from 'react';
import { Navigate } from '../react-router-dom';
function Protected(props) {
    let { component: RouteComponent, path } = props;
    return localStorage.getItem('login') ? <RouteComponent /> :
        <Navigate to={{ pathname: '/login', state: { from: path } }} />
}

export default Protected;
```

12. 配置式路由和懒加载

12.1 src\index.js

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter, Routes, Route, NavLink, Navigate, useRoutes } from './react-router-dom';
const activeStyle = { backgroundColor: 'green' };
const activeClassName = 'active';
const activeNavProps = {
  style: ({ isActive }) => isActive ? activeStyle : {},
  className: ({ isActive }) => isActive ? activeClassName : ''
}
+import routesConfig from './routesConfig';
+const LazyPost = React.lazy(() => import('./components/Post'));
+function App() {
+    let [routes, setRoutes] = React.useState(routesConfig);
+    const addRoute = () => {
+        setRoutes([...routes, {
+            path: '/foo', element: (
+                loading...}>
+
+
+            )
+        }]);
+    }
+    return (
+
+            {useRoutes(routes)}
+            addRoute
+
+    )
+}
ReactDOM.render(

            首页
            用户管理
            个人中心
            foo

+


    , document.getElementById('root'));
```

12.2 src\routesConfig.js

src\routesConfig.js

```
import Home from './components/Home';
import User from './components/User';
import Profile from './components/Profile';
import UserAdd from './components/UserAdd';
import UserDetail from './components/UserDetail';
import UserList from './components/UserList';
import NotFound from './components/NotFound';
import Login from './components/Login';
import Protected from './components/Protected';
const routes = [
    { path: '/', element: <Home /> },
    { path: '/profile', element: <Profile /> },
    {
        path: 'user',
        element: <User />,
        children: [
            { path: 'add', element: <UserAdd /> },
            { path: 'list', element: <UserList /> },
            { path: 'detail/:id', element: <UserDetail /> }
        ]
    },
    { path: '/profile', element: <Protected component={Profile} /> },
    { path: '/login', element: <Login /> },
    { path: '*', element: <NotFound /> }
];
export default routes;
```

12.3 NotFound.js

src\components\NotFound.js

```
import React from 'react';
function NotFound(props) {
    return (
        <div>NotFounddiv>
    )
}
export default NotFound;
```