React全家桶2

课堂目标

1. router使用
2. 整合redux，完成路由守卫逻辑

资源

1. redux
2. react-router
3. react-router

知识要点

react-router

react-router包含3个库，react-router、react-router-dom和react-router-native。react-router提供最基本的路由功能，实际使用的时候我们不会直接安装react-router，而是根据应用运行的环境选择安装react-router-dom（在浏览器中使用）或react-router-native（在rn中使用）。react-router-dom和react-router-native都依赖react-router，所以在安装时，react-router也会自动安装，创建web应用，使用：

安装

```
npm install --save react-router-dom
```

基本使用

react-router中奉行一切皆组件的思想，路由器-**Router**、链接-**Link**、路由-**Route**、独占-**Switch**、重定向-**Redirect**都以组件形式存在

Route渲染优先级：children>component>render

创建RouterPage.js

```
import React, { Component } from "react";
import { BrowserRouter, Link, Route } from "react-router-dom";
import HomePage from "./HomePage";
import UserPage from "./UserPage";

export default class RouterPage extends Component {
  render() {
    return (
      <div>
        <h1>RouterPage</h1>
        <BrowserRouter>
          <nav>
            <Link to="/">首页</Link>
            <Link to="/user">用户中心</Link>
          </nav>
          {/* 根路由要添加exact，实现精确匹配 */}
          <Route exact path="/" component={HomePage} />
          <Route path="/user" component={UserPage} />
        </BrowserRouter>
      </div>
    );
  }
}
```

动态路由

使用:id的形式定义动态路由

定义路由:

```
<Route path="/search/:id" component={Search} />
```

添加导航链接:

```
<Link to={"/search/" + searchId}>搜索</Link>
```

创建Search组件并获取参数:

```
import React, { Component } from "react";
import { BrowserRouter, Link, Route } from "react-router-dom";
import HomePage from "./HomePage";
import UserPage from "./UserPage";

function Search({ match, history, location }) {
  const { id } = match.params;
  return (
    <div>
      <h1>Search: {id}</h1>
    </div>
  );
}

export default class RouterPage extends Component {
  render() {
    const searchId = "1234";
    return (
      <div>
        <h1>RouterPage</h1>
        <BrowserRouter>
          <nav>
            <Link to="/">首页</Link>
            <Link to="/user">用户中心</Link>
            <Link to={"/search/" + searchId}>搜索</Link>
          </nav>
          {/* 根路由要添加exact，实现精确匹配 */}
          <Route exact path="/" component={HomePage} />
          <Route path="/user" component={UserPage} />
          <Route path="/search/:id" component={Search} />
        </BrowserRouter>
      </div>
    );
  }
}
```

嵌套

Route组件嵌套在其他页面组件中就产生了嵌套关系

修改Search，添加新增和详情

```
function Detail() {
  return (
    <div>
```

```
      <h1>Detail</h1>
    </div>
  );
}

function Search({ match, history, location }) {
  const { id } = match.params;
  return (
    <div>
      <h1>Search: {id}</h1>
      <nav>
        <Link to="/search/add">新增</Link>
        <Link to={"/search/detail/" + id}>详情</Link>
      </nav>
      <Route path="/search/add" component={() => <h1>add</h1>} />
      <Route path={"/search/detail/:" + id} component={Detail} />
    </div>
  );
}
```

404页面

设定一个没有path的路由在路由列表最后面，表示一定匹配

```
{/*  添加Switch表示仅匹配一个*/}
<Switch>
  {/* 根路由要添加exact，实现精确匹配 */}
  <Route exact path="/" component={HomePage} />
  <Route path="/user" component={UserPage} />
  <Route path="/search/:id" component={Search} />
  <Route component={() => <h1>404</h1>} />
</Switch>
```

路由守卫

思路：创建高阶组件包装Route使其具有权限判断功能

创建PrivateRoute

```
import React from "react";
import { Route, Redirect } from "react-router-dom";
import LoginPage from "../pages/LoginPage";

export default function PrivateRoute({ component: Cmp, isLogin, ...rest }) {
  return (
    <Route
      {...rest}
      render={props =>
        isLogin ? (
          <Cmp {...props} />
        ) : (
          <Redirect
            to={{
              pathname: "/login",
              state: { redirect: props.location.pathname },
            }}
          />
        )
      }
    />
  );
}
```

创建LoginPage.js

```
import React from "react";
import { Redirect } from "react-router-dom";

export default function LoginPage({ location, isLogin, login }) {
  const redirect = location.state.redirect || "/"; //重定向地址
  if (isLogin) {
    return <Redirect to={redirect} />;
  }
  return (
    <div>
      <p>用户登录</p>
      <br />
      <button onClick={login}>登录</button>
    </div>
  );
}
```

配置路由，RouterPage

```
<Route exact path="/login" component={LoginPage} />
<PrivateRoute path="/user" component={UserPage} />
```

> 给PrivateRoute传递isLogin={true}试试
>
> ```
> <PrivateRoute path="/user" component={UserPage} isLogin={true} />
> ```

> 整合redux，获取和设置登录态，创建./store/index.js

```
import { createStore, combineReducers, applyMiddleware } from "redux";
import thunk from "redux-thunk";

const initialLogin = {
isLogin: false,
name: null,
};
function loginReducer(state = { ...initialLogin }, action) {
switch (action.type) {
 case "getUserInfo":
   return { ...state, isLogin: false };
 case "loginSuccess":
   return { ...state, isLogin: true, name: "xiaoming" };
 case "loginFailure":
   return { ...initialLogin };
 default:
   return state;
}
}

const store = createStore(
combineReducers({ user: loginReducer }),
applyMiddleware(thunk),
);
export default store;
const initialState = { isLogin: false, loading: false };
export default (state = initialState, action) => {
switch (action.type) {
case "requestLogin":
return { isLogin: false, loading: true };
case "loginSuccess":
return { isLogin: true, loading: false };
case "loginFailure":
return { isLogin: false, loading: false };
default:
return state;
}
};
export function login(user) {
return dispatch => {
dispatch({ type: "requestLogin" });
setTimeout(() => {
dispatch({ type: "loginSuccess" });
}, 1000);
};
}
```

连接状态，ReduxTest.js

PrivateRoute.js

```
import React from "react";
import { connect } from "react-redux";
import { Route, Redirect } from "react-router-dom";

function PrivateRoute(props) {
const { component: Cmp, isLogin, ...rest } = props;
return (
 <Route
   {...rest}
   render={props =>
     isLogin ? (
       <Cmp {...props} />
     ) : (
       <Redirect
         to={{
           pathname: "/login",
           state: { redirect: props.location.pathname },
         }}
       />
     )
   }
 />
);
}

export default connect(state => {
return {
 isLogin: state.user.isLogin,
};
})(PrivateRoute);
```

LoginPage.js

```
import React, { Component } from "react";
import { Redirect } from "react-router-dom";
import { connect } from "react-redux";

class LoginPage extends Component {
render() {
 const { location, isLogin, login } = this.props;
 const redirect = location.state.redirect || "/"; //重定向地址
 if (isLogin) {
   return <Redirect to={redirect} />;
 }
 return (
   <div>
     <p>用户登录</p>
```

```
      <br />
      <button onClick={login}>登录</button>
    </div>
  );
 }
 }

 export default connect(
 state => ({ isLogin: state.user.isLogin }),
 {
  login: () => {
    return { type: "loginSuccess" };
  },
 },
 )(LoginPage);
```

UserPage可以再设置一个退出登录

```
import React, { Component } from "react";
import { connect } from "react-redux";

class UserPage extends Component {
  render() {
    const { logout } = this.props;
    return (
      <div>
        <h1>UserPage</h1>
        <button onClick={logout}>退出登录</button>
      </div>
    );
  }
}

export default connect(
  state => state,
  {
    logout: () => ({
      type: "loginFailure",
    }),
  },
)(UserPage);
```

与HashRouter对比：

1. HashRouter最简单，不需要服务器端渲染，靠浏览器的#的来区分path就可以，BrowserRouter需要服务器端对不同的URL返回不同的HTML，后端配置可参考。
2. BrowserRouter使用HTML5历史API（pushState，replaceState和popstate事件），让页面的UI

同步与URL。
3. HashRouter不支持location.key和location.state，动态路由跳转需要通过?传递参数。
4. Hash history 不需要服务器任何配置就可以运行，如果你刚刚入门，那就使用它吧。但是我们不推荐在实际线上环境中用到它，因为每一个 web 应用都应该渴望使用 `browserHistory`。

拓展

react-router秉承一切皆组件，因此实现的核心就是BrowserRouter、Route、Link

实现BrowserRouter

BrowserRouter：历史记录管理对象history初始化及向下传递，location变更监听

创建MyRouterTest.js，首先实现BrowserRouter

```
import { createBrowserHistory } from "history";

const RouterContext = React.createContext();

class BrowserRouter extends Component {
  constructor(props) {
    super(props);

    this.history = createBrowserHistory(this.props);

    this.state = {
      location: this.history.location
    };

    this.unlisten = this.history.listen(location => {
      this.setState({ location });
    });
  }

  componentWillUnmount() {
    if (this.unlisten) this.unlisten();
  }

  render() {
    return (
      <RouterContext.Provider
        children={this.props.children || null}
        value={{
          history: this.history,
          location: this.state.location
        }}
      />
    );
  }
}
```

实现Route

路由配置，匹配检测，内容渲染

```
export function Route(props) {
  const ctx = useContext(RouterContext);
  const { location } = ctx;
  const { path, component, children, render } = props;
  const match = matchPath(location.pathname, props);
  console.log("match", match);
  const matchCurrent = match && match.isExact;
  //const matchCurrent = path === location.pathname;
  const cmpProps = { ...ctx, match };
  console.log("render", render);
  if (matchCurrent && typeof children === "function") {
    return children(cmpProps);
  }
  return (
    <>
      {typeof children === "function" && children(cmpProps)}
      {matchCurrent && component
        ? React.createElement(component, cmpProps)
        : null}
      {matchCurrent && !component && render && render(cmpProps)}
    </>
  );
}
```

> 依赖：matchPath.js
>
> ```
> import pathToRegexp from "path-to-regexp";
>
> const cache = {};
> const cacheLimit = 10000;
> let cacheCount = 0;
>
> function compilePath(path, options) {
> const cacheKey = `${options.end}${options.strict}${options.sensitive}`;
> const pathCache = cache[cacheKey] || (cache[cacheKey] = {});
>
> if (pathCache[path]) return pathCache[path];
>
> const keys = [];
> const regexp = pathToRegexp(path, keys, options);
> ```

```
const result = { regexp, keys };

if (cacheCount < cacheLimit) {
pathCache[path] = result;
cacheCount++;
}

return result;
}

/**
 * Public API for matching a URL pathname to a path.
*/
function matchPath(pathname, options = {}) {
if (typeof options === "string") options = { path: options };

const { path, exact = false, strict = false, sensitive = false } =
options;

const paths = [].concat(path);

return paths.reduce((matched, path) => {
if (!path) return null;
if (matched) return matched;

const { regexp, keys } = compilePath(path, {
end: exact,
strict,
sensitive
});
const match = regexp.exec(pathname);

if (!match) return null;

const [url, ...values] = match;
const isExact = pathname === url;

if (exact && !isExact) return null;

return {
path, // the path used to match
url: path === "/" && url === "" ? "/" : url, // the matched portion of
the URL
isExact, // whether or not we matched exactly
params: keys.reduce((memo, key, index) => {
  memo[key.name] = values[index];
  return memo;
}, {})
};
}, null);
}

export default matchPath;
```

实现Link

Link.js: 跳转链接，处理点击事件

```
export class Link extends Component {
  handleClick(event, history) {
    event.preventDefault();
    history.push(this.props.to);
  }

  render() {
    const { to, children } = this.props;

    return (
      <RouterContext.Consumer>
        {context => {
          return (
            <a
              {...rest}
              onClick={event => this.handleClick(event, context.history)}
              href={to}
            >
              {children}
            </a>
          );
        }}
      </RouterContext.Consumer>
    );
  }
}
```