Router使用及原理解析

课堂目标

1. router使用

2. 整合redux，完成路由守卫逻辑

资源

1. react-router
2. 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, { Component } from "react";
import { Route, Redirect } from "react-router-
dom";
import { connect } from "react-redux";

class PrivateRoute extends Component {
  render() {
    const { path, component, isLogin } =
this.props;
    if (isLogin) {
      return <Route path={path} component=
{component} />;
    } else {
      return (
        <Redirect
          to={{
            pathname: "/login",
            state: { redirect: path },
          }}
        />
      );
    }
```

```
  }
}

export default connect(state => state.user)
(PrivateRoute);
```

创建LoginPage.js

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

class LoginPage extends Component {
  render() {
    const { isLogin, login, location } =
this.props;
    const { redirect = "/" } = location.state
|| {};
    if (isLogin) {
      return <Redirect to={redirect} />;
    }
    return (
      <div>
        <h3>LoginPage</h3>
        <button onClick={login}>login</button>
      </div>
    );
  }
}
```

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

在RouterPage.js配置路由，RouterPage

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

> 整合redux，获取和设置登录态，创建./store/index.js

```
import { createStore, combineReducers } from
"redux";

const initalUserInfo = {
  isLogin: false,
  user: {
    name: "小明",
  },
};
function loginReducer(state = {
...initalUserInfo }, action) {
```

```
  switch (action.type) {
    case "getUserInfo":
      return { ...initalUserInfo };
    case "loginSuccess":
      return { ...state, isLogin: true };
    case "loginFailure":
      return { ...state, isLogin: true };
    default:
      return { ...state };
  }
}
const store = createStore(
  combineReducers({
    user: loginReducer,
  }),
);

export default store;
```

src/index.js

```
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";
import { Provider } from "react-redux";
import store from "./store";

ReactDOM.render(
  <Provider store={store}>
    <App />,
  </Provider>,
  document.getElementById("root"),
);
```

作业：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.user,
  {
    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

创建测试页面MyRouterPage.js，

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

export default class MyRouterPage extends
Component {
  render() {
    return (
      <div>
        <h3>MyRouterPage</h3>
        <BrowserRouter>
          <Link to="/">首页</Link>
          <Link to="/user">用户中心</Link>
```

```
          <Route path="/" exact component=
{HomePage} />
          <Route path="/user" component=
{UserPage} />
        </BrowserRouter>
      </div>
    );
  }
}
```

my-react-router-dom.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 { path, component: Cmp } = props;
  const { location } = ctx;
  let match = path === location.pathname;
  return match ? <Cmp /> : null;
}
```

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>
      );
    }
}
```

回顾

Router使用及原理解析

- 课堂目标
- 资源
- 知识要点
- react-router
  - 安装
  - 基本使用
  - 动态路由
  - 嵌套
  - 404页面
  - 路由守卫
  - 与HashRouter对比：

下节课

React源码解析