React全家桶及原理解析

课堂目标

1. 掌握redux
2. 掌握redux中间件
3. 实现redux、react-redux及其中间件原理
4. 掌握react-router
5. 掌握react-router原理

资源

1. react-router文档
2. 源码

react-router-4

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

创建RouterTest.js

```
import React from "react";
import { BrowserRouter, Link, Route } from "react-router-dom";

function ProductList(props) {
  return <div>ProductList</div>;
}
function ProductMgt(props) {
  return <div>ProductMgt</div>;
}
export default function RouterTest() {
  return (
    <BrowserRouter>
      <nav>
        {/* 导航 */}
        <Link to="/">商品列表</Link>
        <Link to="/management">商品管理</Link>
      </nav>
      <div>
        {/* 直接在组件中定义路由 */}
        {/* 根路由要添加exact, render可以实现条件渲染 */}
        <Route exact path="/" component={ProductList} />
        <Route path="/management" component={ProductMgt} />
      </div>
    </BrowserRouter>
  );
}
```

动态路由

使用:id的形式定义动态路由

定义路由，RouterTest

```
<Route path="/detail/:name" component={Detail} />
```

添加导航链接，ProductList

```
<Link to="/detail/web">web全栈</Link>
```

创建Detail组件并获取参数

```
function Detail({ match, history, location }) {
  console.log(match, history, location);

  return (
    <div>
      ProductMgt
      <p>{match.params.name}</p>
    </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ProductMgt，添加新增和搜索商品

```
function ProductMgt(props) {
  return <div>
      <h3>ProductMgt</h3>
      <Link to="/management/add">新增商品</Link>
      <Link to="/management/search">搜索商品</Link>
      <Route path="/management/add" component={()=><div>add</div>} />
      <Route path="/management/search" component={()=><div>search</div>} />
  </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>
    {/* 首页重定向换成Route方式处理避免影响404 */}
    <Route exact path="/" render={props => <Redirect to="/list" />} />
    {/* <Redirect to="/list"></Redirect> */}
    <Route component={() => <h3>页面不存在</h3>}></Route>
</Switch>
```

路由守卫

创建PrivateRoute组件包装Route使其具有权限判断功能

```
function PrivateRoute({ component: Component, isLogin, ...rest }) {
  // 单独解构出component和isLogin
  // component为渲染目标组件，isLogin通常来自Redux
  // rest为传递给Route的属性
  return (
    <Route
      {...rest}
      render={
        props => // props包含match等信息直接传给目标组件
          isLogin ? ( // 若登陆渲染目标组件
            <Component {...props} />
          ) : ( // 未登录重定向到Login
            <Redirect
              to={{
                pathname: "/login",
                state: { redirect: props.location.pathname } // 重定向地址
              }}
            />
          )
      }
    />
  );
}
```

创建Login

```
function Login({ 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>
      <hr />
      <button onClick={login}>登录</button>
    </div>
  );
}
```

配置路由，ReduxTest

```
<PrivateRoute path="/management" component={ProductMgt} />
<Route path="/login" component={Login} />
```

拓展：react-router实现

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

```
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
        }}
      />
    );
  }
}
```

Route：路由配置，匹配检测，内容渲染

```
import matchPath from "./matchPath";

export default class Route extends Component {
  render() {
    return (
      <RouterContext.Consumer>
        {context => {
          const location = context.location;
          const match = matchPath(location.pathname, this.props)

          // 要传递给下去的属性
          const props = { ...context, match };

          let { children, component, render } = this.props;

          if (children && typeof children === "function") {
            children = children(props);
          }

          return (
            <RouterContext.Provider value={props}>
              {children && React.Children.count(children) > 0
                ? children
                : props.match
                ? component
                  ? React.createElement(component, props)
                  : render
                  ? render(props)
                  : null
                : null}
            </RouterContext.Provider>
          );
        }}
      </RouterContext.Consumer>
    );
  }
}
```

matchPath.js

```
import pathToRegexp from "path-to-regexp";

const cache = {};
const cacheLimit = 10000;
let cacheCount = 0;

// 转换path为正则和关键字数组
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;
}

/**
 * 匹配pathname和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);
    // 转换path为match
    return paths.reduce((matched, path) => {
     if (!path) return null;
     if (matched) return matched;
     // 转换path为正则和占位符数组
     const { regexp, keys } = compilePath(path, {
       end: exact,
       strict,
       sensitive
     });
     // 获得正则匹配数组
     const match = regexp.exec(pathname);

     if (!match) return null;

     // 结构出匹配url和值数组
     const [url, ...values] = match;
     const isExact = pathname === url;

     if (exact && !isExact) return null;

     return {
       path, // 待匹配path
       url: path === "/" && url === "" ? "/" : url, // url匹配部分
       isExact, // 精确匹配
       params: keys.reduce((memo, key, index) => { // 参数
         memo[key.name] = values[index];
         return memo;
       }, {})
     };
    }, null);
}

export default matchPath;
```

Link.js: 跳转链接，处理点击事件

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

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

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

作业

熟练掌握react-router

整合redux，完成路由守卫部分逻辑

深入理解react-router设计理念和实现方式