

React全家桶03

React全家桶03

课堂主题

资源

课堂目标

知识点

Router

BrowserRouter

basename: string

HashRouter

basename: string

BrowserRouter与HashRouter对比:

MemoryRouter

Link

to: string

to: object

replace: bool

others

Redirect

to: string

to: object

Route

Route render methods

path: string

location: object

Router

Switch

location: object

children: node

实现

实现BrowserRouter

实现Route

实现Link

实现Switch

实现Redirect

开始用cra搭建项目

回顾

作业

下节课内容

课堂主题

深入理解Router原理

资源

1. [React中文网](#)

2. [Router-中文](#)

课堂目标

1. Router
2. 搭建cra项目

知识点

Router

BrowserRouter

`<BrowserRouter>` 使用 HTML5 提供的 history API (`pushState`, `replaceState` 和 `popstate` 事件) 来保持 UI 和 URL 的同步。

basename: string

所有URL的base值。如果你的应用程序部署在服务器的子目录，则需要将其设置为子目录。`basename` 的格式是前面有一个/，尾部没有/。

```
<BrowserRouter basename="/kkb">
  <Link to="/user" />
</BrowserRouter>
```

上例中的 `<Link>` 最终将被呈现为：

```
<a href="/kkb/user" />
```

HashRouter

`<HashRouter>` 使用 URL 的 `hash` 部分（即 `window.location.hash`）来保持 UI 和 URL 的同步。

basename: string

同上。

```
<HashRouter basename="/kkb">
  <Link to="/user" />
</HashRouter>
```

上例中的 `<Link>` 最终将被呈现为：

```
<a href="#/kkb/user" />
```

注意：hash history 不支持 `location.key` 和 `location.state`。在以前的版本中，我们曾尝试 shim 这种行为，但是仍有一些边缘问题无法解决。因此任何依赖此行为的代码或插件都将无法正常使用。由于该技术仅用于支持旧浏览器，因此我们鼓励大家使用

`<BrowserHistory>`。

BrowserRouter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 history API（pushState, replaceState和popstate事件），让页面的UI于URL同步。
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`。

MemoryRouter

把 URL 的[历史记录保存在内存](#)中的 `<Router>`（不读取、不写入地址栏）。在测试和非浏览器环境中很有用，如React Native。

```
import { MemoryRouter } from 'react-router-dom';

<MemoryRouter>
  <App />
</MemoryRouter>
```

Link

to: string

一个字符串形式的链接地址，通过 `pathname`、`search` 和 `hash` 属性创建。

```
<Link to='/courses?sort=name' />
```

to: object

一个对象形式的链接地址，可以具有以下任何属性：

- `pathname` - 要链接到的路径
- `search` - 查询参数
- `hash` - URL 中的 hash，例如 `#the-hash`
- `state` - 存储到 `location` 中的额外状态数据

```
<Link to={{
  pathname: '/courses',
  search: '?sort=name',
  hash: '#the-hash',
  state: {
    redirect: '/login'
  }
}} />
```

replace: bool

当设置为 `true` 时，点击链接后将替换历史堆栈中的当前条目，而不是添加新条目。默认为 `false`。

```
<Link to="/courses" replace />
```

others

你还可以传递一些其它属性，例如 `title`、`id` 或 `className` 等。

```
<Link to="/" className="nav" title="a
title">About</Link>
```

Redirect

to: string

要重定向到的 URL，可以是 [path-to-regexp](#) 能够理解的任何有效的 URL 路径。所有要使用的 URL 参数必须由 `from` 提供。

```
<Redirect to="/somewhere/else" />
```

to: object

要重定向到的位置，其中 `pathname` 可以是 [path-to-regexp](#) 能够理解的任何有效的 URL 路径。

```
<Redirect to={{
  pathname: '/login',
  search: '?utm=your+face',
  state: {
    referrer: currentLocation
  }
}} />
```

上例中的 `state` 对象可以在重定向到的组件中通过 `this.props.location.state` 进行访问。而 `referrer` 键（不是特殊名称）将通过路径名 `/login` 指向的登录组件中的 `this.props.location.state.referrer` 进行访问。

Route

`<Route>` 可能是 React Router 中最重要的组件，它可以帮助你理解和学习如何更好的使用 React Router。它最基本的职责是在其 `path` 属性与某个 [location](#) 匹配时呈现一些 UI。

Route render methods

使用 `<Route>` 渲染一些内容有以下三种方式：

- `component`
- `render`: func
- `children`: func

在不同的情况下使用不同的方式。在指定的 `<Route>` 中，你应该只使用其中的一种。

`path: string`

可以是 [path-to-regexp](#) 能够理解的任何有效的 URL 路径。

```
<Route path="/users/:id" component={User} />
```

没有定义 `path` 的 `<Route>` 总是会被匹配。

`location: object`

一般情况下，`<Route>` 尝试将其 `path` 与当前 history location（通常是当前的浏览器 URL）进行匹配。但是，带有不同 pathname 的 location 也可以与之匹配。

当你需要将 `<Route>` 与一个不是当前 `location` 的 location 进行匹配时，会发现这个 api 非常有用。如[过渡动画](#)示例。

如果一个 `<Route>` 被包裹在一个 `<Switch>` 中，并且与 `<Switch>` 的 location 相匹配（或者是当前的 location），那么 `<Route>` 的 `location` 参数将被 `<Switch>` 所使用的 `location` 覆盖。

Router

所有 Router 组件的通用低阶接口。通常情况下，应用程序只会使用其中一个高阶 Router：

- BrowserRouter
- HashRouter
- MemoryRouter
- NativeRouter
- StaticRouter

Switch

用于渲染与[路径匹配的](#)第一个子 `<Route>` 或 `<Redirect>`。

这与仅仅使用一系列 `<Route>` 有何不同？

`<Switch>` 只会渲染一个路由。相反，仅仅定义一系列 `<Route>` 时，每一个与路径匹配的 `<Route>` 都将包含在渲染范围内。考虑如下代码：

```
<Route path="/about" component={About} />
<Route path="/:user" component={User} />
<Route component={NoMatch} />
```

如果 URL 是 `/about`，那么 `<About>`、`<User>` 和 `<NoMatch>` 将全部渲染，因为它们都与路径匹配。这是通过设计，允许我们以很多方式将 `<Route>` 组合成我们的应用程序，例如侧边栏和面包屑、引导标签等。

但是，有时候我们只想选择一个 `<Route>` 来呈现。比如我们在 URL 为 `/about` 时不想匹配 `/:user`（或者显示我们的 404 页面），这该怎么实现呢？以下就是如何使用 `<Switch>` 做到这一点：

```
import { Switch, Route } from 'react-router';

<Switch>
  <Route exact path="/" component={Home} />
  <Route path="/about" component={About} />
  <Route path="/:user" component={User} />
  <Route component={NoMatch} />
</Switch>
```

现在，当我们在 `/about` 路径时，`<Switch>` 将开始寻找匹配的 `<Route>`。我们知道，`<Route path="/about" />` 将会被正确匹配，这时 `<Switch>` 会停止查找匹配项并立即呈现 `<About>`。同样，如果我们在 `/michael` 路径时，那么 `<User>` 会呈现。

这对于动画转换也很有用，因为匹配的 `<Route>` 与前一个渲染位置相同。

```
<Fade>
  <Switch>
    { /* 这里只会渲染一个子元素 */ }
    <Route />
    <Route />
  </Switch>
</Fade>

<Fade>
  <Route />
  <Route />
  { /* 这里总是会渲染两个子元素，也有可能是空渲染，这使得转换更加麻烦 */ }
</Fade>
```

location: object

用于匹配子元素而不是当前history location（通常是当前的浏览器 URL）的 [location](#) 对象。

如下例子，那这里就只匹配首页了。

```
<Switch location={{pathname: "/"}}>
  <Route exact path="/" component=
{HomePage} />
  <Route path="/user" component=
{UserPage} />
  {/* <PrivateRoute path="/user"
component={UserPage} /> */}
  <Route path="/login" component=
{LoginPage} />
  <Route path="/children" children={()
=> <div>children</div>} />
  <Route path="/render" render={() =>
<div>render</div>} />
  <Route path="/search/:id" component=
{searchComponent} />
</Switch>
```

children: node

所有 `<Switch>` 的子元素都应该是 `<Route>` 或 `<Redirect>`。只有第一个匹配当前路径的子元素将被呈现。

`<Route>` 组件使用 `path` 属性进行匹配，而 `<Redirect>` 组件使用它们的 `from` 属性进行匹配。没有 `path` 属性的 `<Route>` 或者没有 `from` 属性的 `<Redirect>` 将始终与当前路径匹配。

当在 `<Switch>` 中包含 `<Redirect>` 时，你可以使用任何 `<Route>` 拥有的路径匹配属性：`path`、`exact` 和 `strict`。`from` 只是 `path` 的别名。

如果给 `<Switch>` 提供一个 `location` 属性，它将覆盖匹配的子元素上的 `location` 属性。

```
<Switch>
  <Route exact path="/" component={Home} />
  <Route path="/users" component={Users} />
  <Redirect from="/accounts" to="/users" />
  <Route component={NoMatch} />
</Switch>
```

实现

测试代码：

app.js

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

import {BrowserRouter as Router} from "../k-
react-router-dom/BrowserRouter";
import Route from "../k-react-router-dom/Route";
import Link from "../k-react-router-dom/Link";
```

```

import Switch from "./k-react-router-dom/Switch";

import HomePage from "./pages/HomePage";
import UserPage from "./pages/UserPage";
import LoginPage from "./pages/LoginPage";
import PrivateRoute from
"./pages/PrivateRoute";

function App() {
  return (
    <div className="App">
      <Router>
        <Link to="/">首页</Link>
        <Link to="/user">
          用户中心
        </Link>
        <Link to="/login">登录</Link>
        <Link to="/children">children</Link>
        <Link to="/render">render</Link>
        <Link to="/search/123">搜索-动态路由</Link>

        { /* <Switch location={{pathname: "/"}}>
          */ }

        <Switch>
          <Route exact path="/" component=
{HomePage} />

```

```

        { /* <Route path="/user" component=
{UserPage} /> */}
        <PrivateRoute path="/user" component=
{UserPage} />
        <Route path="/login" component=
{LoginPage} />
        <Route path="/children" children={()
=> <div>children</div>} />
        <Route path="/render" render={() =>
<div>render</div>} />
        <Route path="/search/:id" component=
{searchComponent} />
    </Switch>
  </Router>
</div>
);
}

export default App;

function searchComponent(props) {
  const {id} = props.match.params;
  console.log("searchComponent-props", props);
  //sy-log
  return (
    <div>
      <div>searchComponent-{id}</div>
      <Link to={"/search/" + id + "/detail"}>详
情-嵌套路由</Link>

```



```

        <Route path={"/search/:id/detail"}
component={DetailComponent} />
    </div>
    );
}

function DetailComponent(props) {
    const {id} = props.match.params;
    console.log("DetailComponent-props", props);
    //sy-log
    return <div>DetailComponent</div>;
}

```

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

BrowserRouter

```

import React, {Component} from "react";
import {createBrowserHistory} from "history";
import matchPath from "../matchPath";

const RouterContext = React.createContext();

export class BrowserRouter extends Component {
    constructor(props) {

```

```

    super(props);
    this.state = {};
    this.history = createBrowserHistory();
    this.state = {
      location: this.history.location
    };
    this.unlisten =
this.history.listen(location => {
    this.setState({
      location: location
    });
  });
}

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

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

```

```
}
```

实现Route

路由配置, 匹配检测, 内容渲染

```
export class Route extends Component {
  render() {
    return (
      <RouterContext.Consumer>
        {context => {
          // 优先从props中取值
          const location = this.props.location
          || context.location;
          // 优先从props中取值计算
          const match =
            this.props.computedMatch
              ? this.props.computedMatch
              : this.props.path
              ? matchPath(location.pathname,
                this.props)
              : context.match;
          const props = {
            ...context,
            location,
            match
          };
        }}
    );
  }
}
```

```

        let {path, children, component,
render} = this.props;
        // match 渲染这三者之一: children
        component render或者null
        // 不match, 渲染children 或者 null
        return (
            <RouterContext.Provider value=
{props}>
                {match
                    ? children
                    ? typeof children ===
"function"
                        ? children(props)
                        : children
                    : component
                    ?
React.createElement(component, props)
                        : render
                    ? render(props)
                        : null
                    : typeof children ===
"function"
                        ? children(props)
                        : null}
                </RouterContext.Provider>
            );
        }}
    </RouterContext.Consumer>
);

```

如果children是func
则执行func

```
}  
}
```

实现Link

Link.js: 跳转链接，处理点击事件

```
export class Link extends Component {  
  static contextType = RouterContext;  
  handleClick = event => {  
    event.preventDefault();  
    this.context.history.push(this.props.to);  
  };  
  render() {  
    const {to} = this.props;  
    return (  
      <a href={to} onClick={this.handleClick}>  
        {this.props.children}  
      </a>  
    );  
  }  
}
```

实现Switch

```
export class Switch extends Component {
```

```

render() {
  return (
    <RouterContext.Consumer>
      {context => {
        const location = context.location;
        let element, match;

        React.Children.forEach(this.props.children,
child => {
          if (match == null &&
React.isValidElement(child)) {
            element = child;
            const path = child.props.path;
            match = path
              ? matchPath(location.pathname,
{...child.props, path})
              : context.match;
          }
        });
        return match ?
React.cloneElement(element, {location}) : null;
      }}
    </RouterContext.Consumer>
  );
}
}

```

实现Redirect

```
import React, {Component} from "react";
import {RouterContext} from "../RouterContext";

export default class Redirect extends Component
{
  render() {
    return (
      <RouterContext.Consumer>
        {context => {
          const {history} = context;
          const {to} = this.props;
          // history.push(location)
          return <Lifecycle onMount={() =>
            history.replace(to)} />;
        }}
      </RouterContext.Consumer>
    );
  }
}

class Lifecycle extends Component {
  componentDidMount() {
    if (this.props.onMount) {
      this.props.onMount.call(this, this);
    }
  }
  render() {
```

```
    return null;  
  }  
}
```

开始用cra搭建项目

参考github：（git搜bubucuo，在code下）。

<https://github.com/bubucuo/code/tree/master>

考虑用到技术：

1. redux、react-redux管理数据，注意页面数据存储。
2. BrowserRouter管理路由，注意权限问题。
3. 建议不要使用antd等框架，比如说实现登录，自己来实现，UI丑点没事，重点实现功能。

回顾

React全家桶03

课堂主题

资源

课堂目标

知识点

Router

BrowserRouter

basename: string

HashRouter

 basename: string

BrowserRouter与HashRouter对比:

MemoryRouter

Link

 to: string

 to: object

 replace: bool

others

Redirect

 to: string

 to: object

Route

 Route render methods

 path: string

 location: object

Router

Switch

 location: object

 children: node

实现

 实现BrowserRouter

 实现Route

 实现Link

 实现Switch

 实现Redirect

开始用cra搭建项目

回顾

作业

作业

1. Router代码自己写至少一遍。
2. 用cra搭建个项目吧，建议不要使用antd等框架，比如说实现登录，自己来实现，UI丑点没事，重点实现功能。

下节课内容

redux-saga, umi。