回顾

1. redux
   - store创建
   - 注册
   - connect(state=>({xx,bb}), {login})(Comp)
2. react-router
   - BrowserRouter
   - Route
3. 原理回顾

拓展

- react-router原理，源码
  - BrowserRouter：history初始化及向下传递，location变更监听

```
import React, { Component } from "react";
import { createBrowserHistory as createHistory } from "history";

export const RouterContext = React.createContext();

export default class BrowserRouter extends Component {
  static computeRootMatch(pathname) {
    return { path: "/", url: "/", params: {}, isExact: pathname === "/" };
  }

  constructor(props) {
    super(props);

    this.history = createHistory(this.props);

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

    this._isMounted = false;
    this._pendingLocation = null;

    this.unlisten = this.history.listen(location => {
      if (this._isMounted) {
        this.setState({ location });
      } else {
        this._pendingLocation = location;
      }
    });
  }
```

```
  componentDidMount() {
    this._isMounted = true;

    if (this._pendingLocation) {
      this.setState({ location: this._pendingLocation });
    }
  }

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

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

- Route：路由配置，匹配检测，内容渲染

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

export default class Route extends Component {
  render() {
    return (
      <RouterContext.Consumer>
        {context => {
          const location = this.props.location || context.location;
          const match = this.props.computedMatch
            ? this.props.computedMatch // <Switch> already computed the match
for us
            : this.props.path
            ? matchPath(location.pathname, this.props)
            : context.match;

          const props = { ...context, location, match };

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

          // 若未传递children属性，则默认为null
          if (Array.isArray(children) && children.length === 0) {
```

```
            children = null;
          }

          if (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>
    );
  }
}
```

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;
```

- Link.js: 跳转链接，处理点击事件

```
import React from "react";
import { RouterContext } from "./BrowserRouter";
import { createLocation } from "history";

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

```
  }

  render() {
    const { to, ...rest } = this.props; // eslint-disable-line no-unused-vars

    return (
      <RouterContext.Consumer>
        {context => {
          const location =
            typeof to === "string"
              ? createLocation(to, null, null, context.location)
              : to;
          const href = location ? context.history.createHref(location) : "";

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

export default Link;
```

知识点

1. redux-saga
   - redux-thunk可选方案
   - 优点：利用generator，强大，action依然是对象，易管理，执行、测试、失败处理
   - 使用：
     - 安装 `npm install --save redux-saga`
     - 创建清单sagas.js

```
import { call, put, takeEvery } from "redux-saga/effects";

// worker Saga
function* login(action) {}

function* mySaga() {
  yield takeEvery("login", login);
}

export default mySaga;
```

- 注册

```
import createSagaMiddleware from "redux-saga";
import mySaga from "./sagas";

// 1.创建saga中间件并注册
const sagaMiddleware = createSagaMiddleware();

const store = createStore(
  combineReducers({ user }),
  applyMiddleware(logger, sagaMiddleware)
);
// 2.中间件运行saga
sagaMiddleware.run(mySaga);
export default store;
```

2. generator

- 基本概念：流程控制语句

```
function* g(){
    yield 'a';
    return 'b';
}
const gen = g();
gen.next(); // {value:'a',done:false}
gen.next(); // {value:'b',done:true}
```

- 传参

```
function* g(x){
    let y = yield x;
    yield y;
}
const gen = g(1);
gen.next(2); // {value:1,done:false}
gen.next(); // {value:2,done:true}
```

- 异步

```
function* g(a){
    let b = yield asyncFetch(a);
    yield asyncFetch(b);
}
function asyncFetch(x){
    return new Promise((resolve) => {
        resolve(x*x)
    })
}
const gen = g(2);
gen.next().value
   .then(r=>gen.next(r).value)
   .then(r=>console.log(r))
```

3. umi + dva
   - 安装：npm install umi -D
   - 自动生成路由：`umi g page index`
   - 起服务：`umi dev`
   - 动态路由：以$开头的文件或目录

```
// 创建users/$id.js，内容和其他页面相同，显示一下传参
export default function(props) {
  return (
    <div>
      <h1>user id: {props.match.params.id}</h1>
    </div>
  );
}
```

   - 嵌套路由：目录下创建_layout

```
// 创建父组件 umi g page users/_layout
export default function(props) {
  return (
    <div>
      <h1>Page _layout</h1>
      <div>{props.children}</div>
    </div>
  )
}
// 创建兄弟组件 umi g page users/index
```

- 页面跳转

```
import Link from "umi/link";
import router from "umi/router";

<Link to={`/users/${u.id}`}>{u.name}</Link>
<li key={u.id} onClick={()=>router.push(`/users/${u.id}`)}>{u.name}</li>
```

- 配置路由：业务复杂后仍需配置路由

```
// 创建~/config/config.js
export default {
  routes: [
    { path: "/", component: "./index" },
    {
      path: "/users",
      component: "./users/_layout",
      routes: [
        { path: "/users/", component: "./users/index" },
        { path: "/users/:id", component: "./users/$id" }
      ]
    }
  ]
};
```

- 404页面：添加不带path的路由配置项：`{component: './NotFound'}`

- 权限路由

```
{
    path: "/about",
    component: "./about",
    Routes: ["./routes/PrivateRoute.js"] // 这里相对根目录，文件名后缀不能少
}
```

- **引入antd**
  - 添加antd：`npm install antd -S`
  - 添加 umi-plugin-react: `npm install umi-plugin-react -D`
  - 修改~/config/config.js

```
plugins: [
  ['umi-plugin-react', {
    antd: true
  }],
],
```

- 数据流管理dva
  - 配置

```
export default {
  plugins: [
    ['umi-plugin-react', {
      antd: true,
      dva: true,
    }],
  ],
  // ...
}
```

  - 创建model

```
export default {
  namespace: 'goods', // model的命名空间，区分多个model
  state: [{ title: "web全栈" },{ title: "java架构师" }], // 初始状态
  effects:{}, // 异步操作
  reducers: { // 更新状态 }
}
```

  - 使用状态

```
import { connect } from "dva";

@connect(
  state => ({
    goodsList: state.goods // 获取指定命名空间的模型状态
  })
)
class Goods extends Component {}
```