项目实战01

课堂目标

1. 掌握生成器函数 - generator
2. 掌握redux异步方案 - redux-saga

资源

1. redux-saga：中文、英文
2. generator

知识点

Generator

Generator 函数是 ES6 提供的一种异步编程解决方案，语法行为与传统函数完全不同，详细参考文章。

1. function关键字与函数名之间有一个*;
2. 函数体内部使用yield表达式，定义不同的内部状态。
3. yield表达式只能在 Generator 函数里使用，在其他地方会报错。

```
function* helloWorldGenerator() {
  yield 'hello';
  yield 'world';
  return 'ending';
}

var hw = helloWorldGenerator();

//执行
console.log(hw.next());
console.log(hw.next());
console.log(hw.next());
console.log(hw.next());
```

4. `yield`表达式后面的表达式，只有当调用`next`方法、内部指针指向该语句时才会执行，因此等于为 JavaScript 提供了手动的“惰性求值”（Lazy Evaluation）的语法功能。

```
var a = 0;
function* fun() {
  let aa = yield (a = 1 + 1);
  return aa;
}

console.log("fun0", a);
let b = fun();
console.log("fun", b.next());//注释下这句试试，比较下前后a的值
console.log("fun1", a);
```

由于 Generator 函数返回的遍历器对象，只有调用`next`方法才会遍历下一个内部状态，所以其实提供了一种可以暂停执行的函数。`yield`表达式就是暂停标志。

手动搭建移动端项目

管理数据redux

路由管理react-router-dom

异步操作thunk或者saga

Routes

```
import React from "react";

import {BrowserRouter as Router, Route, Switch,
Link} from "react-router-dom";
import PrivateRoute from "./PrivateRoute";
import HomePage from "../pages/HomePage/";
import UserPage from "../pages/UserPage/";
import LoginPage from "../pages/LoginPage/";
import _404 from "../pages/_404/";

export default function Routes(props) {
  return (
```

```
    <Router>
      <Switch>
        <Route path="/" exact component=
{HomePage} />
        {/* <Route path="/user"  component=
{UserPage} /> */}
        <Route path="/login" component=
{LoginPage} />
        <PrivateRoute path="/user">
          <UserPage />
        </PrivateRoute>
        <Route component={_404} />
      </Switch>
    </Router>
  );
}
```

BasicLayout

```
import React, {Component} from "react";
import BottomNav from
"../../components/BottomNav";
import TopBar from "../../components/TopBar";
import classnames from "classnames";
import "./index.scss";
```

```
export default class BasicLayout extends
Component {
  componentDidMount() {
    const {
      title = "商城",
      shortIcon = "https://store-images.s-
microsoft.com/image/apps.64108.9007199266248398
.f50070aa-ca14-4881-9e29-fb874435dc3d.a620dd2f-
083d-4523-bdd5-d50a527956d4"
    } = this.props;
    document.title = title;
    shortIcon &&
(document.getElementById("shortIcon").href =
shortIcon);
  }
  render() {
    const {
      children,
      showTopBar = true,
      title = "商城",
      _className
    } = this.props;
    return (
      <div className={classnames("basicLayout",
_className)}>
        {showTopBar && <TopBar title={title}
/>}
        <article>{children}</article>
        <BottomNav />
```

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

PrivateRoute

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

export default connect(
  //mapStateToProps
  ({user}) => ({
    isLogin: user.isLogin
  })
)(function PrivateRoute({children, isLogin,
...rest}) {
  return (
    <Route
      {...rest}
      render={({location}) =>
        isLogin ? (
          children
        ) : (
```

```
            <Redirect
              to={{pathname: "/login", state:
{redirect: location.pathname}}}
            />
          )
        }
      />
    );
});
```

LoginPage

```
import React, {Component} from "react";
import {Redirect} from "react-router-dom";
import {connect} from "react-redux";
import BasicLayout from
"../../layout/BasicLayout/";
// import {loginAction} from
"../../action/login";
import "./index.scss";

export default connect(({user}) => ({user}), {
  // login: userInfo => ({type:
"LOGIN_SUCCESS", payload: userInfo})
  // login: userInfo => dispatch => {
  //   loginAction(dispatch, userInfo);
  //   // dispatch({type: "LOGIN_REQUEST"});
```

```
  //   // setTimeout(() => {
  //   //   dispatch({type: "LOGIN_SUCCESS",
payload: userInfo});
  //   // }, 1000);
  // }
  // saga
  login: userInfo => ({type: "loginSaga",
payload: userInfo})
})(
  class LoginPage extends Component {
    constructor(props) {
      super(props);
      this.state = {name: ""};
    }
    render() {
      const {login, user, location} =
this.props;
      const {isLogin, loading, err, tip} =
user;
      if (isLogin) {
        const {redirect = "/"} = location.state
|| {};
        return <Redirect to={redirect} />;
      }
      const {name} = this.state;
      return (
        <BasicLayout title="登录"
_className="loginPage">
          <h3>LoginPage</h3>
```

```
            <input
              type="text"
              value={name}
              onChange={event =>
this.setState({name: event.target.value})}
            />
            <p className="red">{err.msg}</p>
            <button onClick={() =>
login({name})}>
              {loading ? "登录中..." : "登录"}
            </button>
            <p className="green">{tip.msg}</p>
          </BasicLayout>
        );
      }
    }
);
```

action/login.js

async 函数是什么？一句话，它就是 Generator 函数的语法糖。

async 函数的实现原理，就是将 Generator 函数和自动执行器，包装在一个函数里。

```
import LoginService from "../service/login";
```

```
// async await
// const res1 =  ajax1
// ajax2(res1)

//async原理也是generator，但是比generator简单
export async function loginAction(dispatch,
userInfo) {
  dispatch({type: "LOGIN_REQUEST"});
  const res1 = await login(dispatch, userInfo);
  getMoreUserInfo(dispatch, res1);
}

// export function loginAction(dispatch,
userInfo) {
//   dispatch({type: "LOGIN_REQUEST"});
//   // setTimeout(() => {
//   //   dispatch({type: "LOGIN_SUCCESS",
payload: userInfo});
//   // }, 1000);
//   login(dispatch, userInfo);
// }

function login(dispatch, userInfo) {
  return LoginService.login(userInfo).then(
    res => {
      return res;
      // dispatch({type: "LOGIN_SUCCESS",
payload: res});
      // getMoreUserInfo(dispatch, res);
```

```
    },
    err => {
      dispatch({type: "LOGIN_FAILURE", payload:
err});
    }
  );
}

function getMoreUserInfo(dispatch, userInfo) {
  return
LoginService.getMoreUserInfo(userInfo).then(
    res => {
      dispatch({type: "LOGIN_SUCCESS", payload:
{...userInfo, ...res}});
      return res;
    },
    err => {
      dispatch({type: "LOGIN_FAILURE", payload:
err});
    }
  );
}
```

redux-saga

- 概述：`redux-saga` 是一个用于管理应用程序 Side Effect（副作用，例如异步获取数据，访问浏览器缓存

等）的 library，它的目标是让副作用管理更容易，执行更高效，测试更简单，在处理故障时更容易。

- 地址：https://github.com/redux-saga/redux-saga
- 安装：**npm install --save redux-saga**
- 使用：用户登录

在 `redux-saga` 的世界里，Sagas 都用 Generator 函数实现。我们从 Generator 里 yield 纯 JavaScript 对象以表达 Saga 逻辑。 我们称呼那些对象为 *Effect*。

你可以使用 `redux-saga/effects` 包里提供的函数来创建 Effect。

effects

effect 是一个 javascript 对象，里面包含描述副作用的信息，可以通过 yield 传达给 sagaMiddleware 执行。

在 redux-saga 世界里，所有的 effect 都必须被 yield 才会执行，所以有人写了 eslint-plugin-redux-saga 来检查是否每个 Effect 都被 yield。并且原则上来说，所有的 yield 后面也只能跟effect，以保证代码的易测性。

put

作用和 redux 中的 dispatch 相同。

```
yield put({ type: "loginSuccess");
```

call与fork：阻塞调用和无阻塞调用

redux-saga 可以用 fork 和 call 来调用子 saga，其中 fork 是无阻塞型调用，call 是阻塞型调用，即call是有阻塞地调用 saga 或者返回 promise 的函数。

take

等待 redux dispatch 匹配某个 pattern 的 action 。

```
function* loginSaga(props) {
    // yield takeEvery("login", loginHandle);
    // 等同于
    const action = yield take("loginSaga");
    yield fork(loginHandle, action);
}
```

takeEvery

`takeEvery` 可以让多个 `saga` 任务并行被 fork 执行。

```
import {fork, take} from "redux-saga/effects"

const takeEvery = (pattern, saga, ...args) =>
fork(function*() {
  while (true) {
    const action = yield take(pattern)
    yield fork(saga, ...args.concat(action))
  }
})
```

redux-saga 使用了 ES6 的 Generator 功能，让异步的流程更易于读取，写入和测试。（*如果你还不熟悉的话，这里有一些介绍性的链接*）通过这样的方式，这些异步的流程看起来就像是标准同步的 Javascript 代码。（有点像 `async`/`await`，但 Generator 还有一些更棒而且我们也需要的功能）。

不同于 redux-thunk，**你不会再遇到回调地狱了**，你可以很容易地测试异步流程并保持你的 action 是干净的，因此我们可以说**redux-saga更擅长解决复杂异步这样的场景，也更便于测试。**

saga的方式实现路由守卫

1. 创建一个./action/userSaga.js处理用户登录请求

call： 调用异步操作

put：状态更新

takeEvery：做saga监听

action/loginSaga.js

```
// 调用异步操作 call、
// 状态更新 （dispatch） put
// 做监听 take

import {
  call,
  fork,
  put,
  take
  // takeEvery
} from "redux-saga/effects";
import LoginService from "../service/login";

// worker saga
function* loginHandle(action) {
  // 调用异步操作 call
  yield put({type: "LOGIN_REQUEST"});
  try {
    const res1 = yield call(LoginService.login,
action.payload);
```

```
    const res2 = yield
call(LoginService.getMoreUserInfo, res1);
    yield put({type: "LOGIN_SUCCESS", payload:
{...res1, ...res2}});
  } catch (err) {
    yield put({type: "LOGIN_FAILURE", payload:
err});
  }
}

// watcher saga

function* loginSaga() {
  // while (true) {
  // const action = yield take("loginSaga");
  // call 是一个会阻塞的 Effect。即 Generator 在调
用结束之前不能执行或处理任何其他事情。
  // yield call(loginHandle, action);
  // console.log("loginSaga-res", action);
//sy-log

  // fork 是无阻塞型调用,
  // 当我们 fork 一个 任务，任务会在后台启动，调用者也
可以继续它自己的流程，而不用等待被 fork 的任务结束。
  // yield fork(loginHandle, action);
  // console.log("loginSaga-res", action);
//sy-log
  // }
  yield takeEvery("loginSaga", loginHandle);
```

```
}

export default loginSaga;

const takeEvery = (pattern, saga, ...args) =>
  fork(function*() {
    while (true) {
      const action = yield take(pattern);
      yield fork(saga, ...args.concat(action));
    }
  });
```

store/index.js

注册redux-saga

```
import {createStore, combineReducers,
applyMiddleware} from "redux";
import thunk from "redux-thunk";
import createSagaMiddleware from "redux-saga";
import {userReducer} from "./userReducer";
import loginSaga from "../action/loginSaga";

const sagaMiddleware = createSagaMiddleware();

const store = createStore(
  combineReducers({user: userReducer}),
  // applyMiddleware(thunk)
```

```
  applyMiddleware(sagaMiddleware)
);
sagaMiddleware.run(loginSaga);

export default store;
```

LoginPage.js

```
export default connect(({user}) => ({user}), {
  login: userInfo => ({type: "loginSaga",
payload: userInfo})
})(
  class LoginPage extends Component {
  //...
  }
);
```

> redux-saga基于generator实现，使用前搞清楚generator相当重要

当有多个saga的时候，rootSaga.js

```
import {all} from "redux-saga/effects";
import userSaga from "./userSaga";

export default function* rootSaga() {
  yield all([userSaga()]);
}
```

store/index.js中引用改成rootSaga即可：

```
...
sagaMiddleware.run(rootSaga);
```

回顾

项目实战01

- 课堂目标
- 资源
- 知识点
  - Generator
  - 手动搭建移动端项目
    - Routes
    - BasicLayout
    - PrivateRoute
    - LoginPage
    - action/login.js
  - redux-saga
    - effects

作业

丰富自己的项目，预习umi、dva、antd。

下节课内容

掌握umi、dva、antd，用框架写React项目。