项目实战

课堂目标

1. 掌握企业级应用框架 - umi
2. 掌握数据流方案 - dva
3. 掌握生成器函数 - generator

4. 掌握redux异步方案 - redux-saga

资源

1. umi
2. dva
3. Antd Pro
4. redux-saga：中文、英文
5. generator
6. rc-notification

知识要点

1. generator用法
2. redux-saga用法
3. umi用法

起步

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());
```

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。

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

src/App

```
import React, { Component } from "react";
import { BrowserRouter, Link, Route, Switch }
from "react-router-dom";
import HomePage from "./pages/HomePage";
import UserPage from "./pages/UserPage";
import LoginPage from "./pages/LoginPage";
import PrivateRoute from
"./pages/PrivateRoute";
import { connect } from "react-redux";

class App extends Component {
  render() {
    const { userName } = this.props;
    return (
      <div className="App">
        <BrowserRouter>
          <Link to="/">首页</Link>
          <Link to="/user">个人中心</Link>
```

```
          <Link to={userName ? "/user" :
"/login"}>{userName || "登录"}</Link>

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

export default connect(state => ({
  userName: state.userName
}))(App);
```

创建store/index.js

```
import { createStore, applyMiddleware } from
"redux";
import thunk from "redux-thunk";
```

```
const loginInfo = {
  isLogin: false,
  loading: false,
  userName: ""
};

function loginReducer(state = { ...loginInfo },
action) {
  console.log("action", action.payload);
  switch (action.type) {
    case "loginRequest":
      return { ...state, ...loginInfo, loading:
true };
    case "loginSuccess":
      return { ...state, isLogin: true,
loading: false, ...action.payload };
    case "loginFailure":
      return { ...state, ...loginInfo,
...action.payload };
    default:
      return state;
  }
}
const store = createStore(loginReducer,
applyMiddleware(thunk));

export default store;
```

登录页面pages/LoginPage.js

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

class LoginPage extends Component {
  constructor(props) {
    super(props);
    this.state = { userName: "" };
  }
  render() {
    const { isLogin, location, login, loading,
err } = this.props;
    const { redirect = "/" } = location.state
|| {};
    if (isLogin) {
      return <Redirect to={redirect} />;
    }
    return (
      <div>
        <h3>LoginPage</h3>
        <input
          value={this.state.userName}
          onChange={event => this.setState({
userName: event.target.value })}
        />
```

```
        <button onClick={() =>
login(this.state.userName)}>
          {loading ? "loading..." : "login"}
        </button>
        <p>{err}</p>
      </div>
    );
  }
}

export default connect(
  state => ({
    isLogin: state.isLogin,
    loading: state.loading,
    err: state.err
  }),
  {
    login: userName => dispatch => {
      dispatch({ type: "loginRequest" });
      setTimeout(() => {
        dispatch({ type: "loginSuccess",
payload: { userName } });
      }, 1000);
    }
  }
)(LoginPage);
```

路由守卫/pages/PrivatePage.js：

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

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

export default connect(
  state => ({
    isLogin: state.isLogin
```

```
  }),
  {}
)(PrivateRoute);
```

用saga的方式实现：

1. 创建一个./store/mySaga.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

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

// 模拟登录接口
const UserService = {
  login(userName) {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        if (userName === "小明") {
          resolve({ userName: "小明" });
        } else {
          reject({ err: "用户名或密码错误" });
        }
      }, 1000);
    });
  }
```

```
};

//worker saga
function* loginHandle(action) {
  try {
    yield put({ type: "loginRequest" });
    //登录
    const res = yield call(UserService.login,
action.payload.userName);
    console.log("res", res);
    yield put({ type: "loginSuccess", payload:
{ ...res } });
  } catch (err) {
    yield put({ type: "loginFailure", payload:
{ ...err } });
  }
}

//watcher saga
function* mySaga(props) {
  yield takeEvery("login", loginHandle);
}

export default mySaga;
```

2. 注册redux-saga，./store/index.js

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

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

const sagaMiddleware = createSagaMiddleware();

const loginInfo = {
  isLogin: false,
  loading: false,
  userName: ""
};

function loginReducer(state = { ...loginInfo },
action) {
  switch (action.type) {
    case "loginRequest":
      return { ...state, ...loginInfo, loading:
true };
    case "loginSuccess":
      return { ...state, isLogin: true,
loading: false, ...action.payload };
    case "loginFailure":
      return { ...state, ...loginInfo,
...action.payload };
    default:
      return state;
  }
}
```

```
const store = createStore(loginReducer,
applyMiddleware(sagaMiddleware));

sagaMiddleware.run(mySaga);

export default store;
```

4. 测试，LoginPage.js

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

class LoginPage extends Component {
  constructor(props) {
    super(props);
    this.state = { userName: "" };
  }
  render() {
    const { isLogin, location, login, loading,
err } = this.props;
    const { redirect = "/" } = location.state
|| {};
    if (isLogin) {
      return <Redirect to={redirect} />;
    }
    return (
      <div>
```

```
        <h3>LoginPage</h3>
        <input
          value={this.state.userName}
          onChange={event => this.setState({
userName: event.target.value })}
        />
        <button onClick={() =>
login(this.state.userName)}>
          {loading ? "loading..." : "login"}
        </button>
        <p>{err}</p>
      </div>
    );
  }
}

export default connect(
  state => ({
    isLogin: state.isLogin,
    loading: state.loading,
    err: state.err
  }),
  {
    login: userName => ({ type: "login",
payload: { userName } })
  }
)(LoginPage);
```

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

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**。

umi

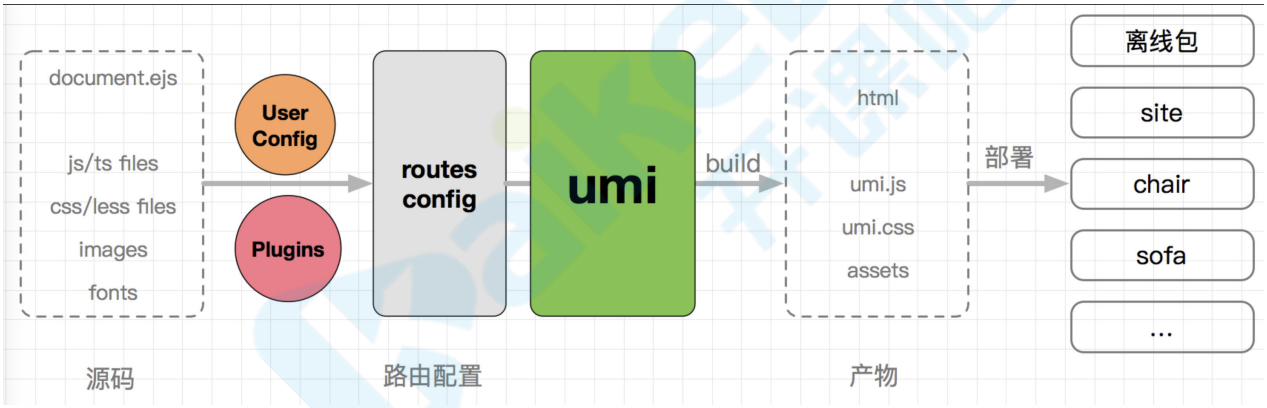
umi，中文可发音为乌米，是一个可插拔的企业级 react 应用框架。

why umi

- 📦 **开箱即用**，内置 react、react-router 等
- 🏈 **类 next.js 且功能完备的路由约定**，同时支持配置的路由方式
- 🎉 **完善的插件体系**，覆盖从源码到构建产物的每个生命周

期

- 🚀 **高性能**，通过插件支持 PWA、以路由为单元的 code splitting 等
- 💈 **支持静态页面导出**，适配各种环境，比如中台业务、无线业务、egg、支付宝钱包、云凤蝶等
- 🚄 **开发启动快**，支持一键开启 dll 和 hard-source-webpack-plugin 等
- 🐠 **一键兼容到 IE9**，基于 umi-plugin-polyfills
- 🍁 **完善的 TypeScript 支持**，包括 d.ts 定义和 umi test
- 🌴 **与 dva 数据流的深入融合**，支持 duck directory、model 的自动加载、code splitting 等等



它和 dva、roadhog 是什么关系?

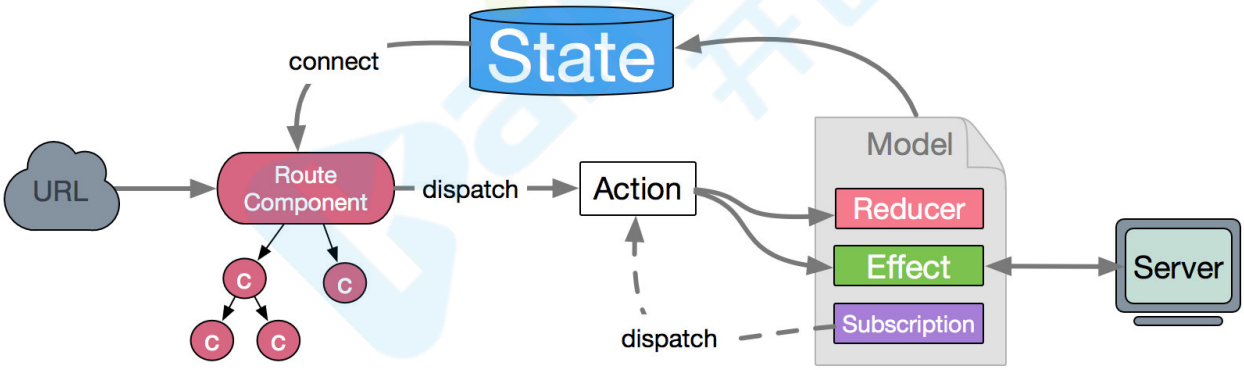
- roadhog 是基于 webpack 的封装工具，目的是简化 webpack 的配置
- umi 可以简单地理解为 roadhog + 路由，思路类似 next.js/nuxt.js，辅以一套插件机制，目的是通过框架的

方式简化 React 开发

- dva 目前是纯粹的数据流，和 umi 以及 roadhog 之间并没有相互的依赖关系，可以分开使用也可以一起使用，个人觉得 umi + dva 是比较搭的

dva

dva 首先是一个基于 redux 和 redux-saga 的数据流方案，然后为了简化开发体验，dva 还额外内置了 react-router 和 fetch，所以也可以理解为一个轻量级的应用框架。



dva+umi 的约定

- src 源码
  - pages页面
  - components 组件
  - layout布局

- model
- config 配置
- mock 数据模拟
- test测试等

```
.
├── dist/                          // 默认的 build 输出目录
├── mock/                          // mock 文件所在目录，基于 express
├── config/
    ├── config.js                  // umi 配置，同 .umirc.js，二选一
└── src/                           // 源码目录，可选
    ├── layouts/index.js           // 全局布局
    ├── pages/                     // 页面目录，里面的文件即路由
        ├── .umi/                  // dev 临时目录，需添加到 .gitignore
        ├── .umi-production/       // build 临时目录，会自动删除
        ├── document.ejs           // HTML 模板
        ├── 404.js                 // 404 页面
        ├── page1.js               // 页面 1，任意命名，导出 react 组件
        ├── page1.test.js          // 用例文件，umi test 会匹配所有 .test.js 和 .e2e.js 结尾
        └── page2.js               // 页面 2，任意命名
    ├── global.css                 // 约定的全局样式文件，自动引入，也可以用 global.less
    ├── global.js                  // 可以在这里加入 polyfill
    ├── app.js                     // 运行时配置文件
├── .umirc.js                      // umi 配置，同 config/config.js，二选一
├── .env                           // 环境变量
└── package.json
```

安装

环境要求： node版本>=8.10

```
antd-pro安装：
新建立一个空文件夹：mkdir lesson8-umi
进入文件夹：cd lesson8-umi
创建：yarn create umi
选择ant-design-pro
选择Javascript
安装依赖： yarn
启动：yarn start或者umi dev
```

其他例子： 如umi-antd-mobile等

Umi基本使用

建立pages下面的单页面about：

```
umi g page about
```

建立文件夹channel(默认是css)：

```
umi g page channel/index --less
```

import router from 'umi/router' 跳转
router.push('/user/2')

起服务看效果

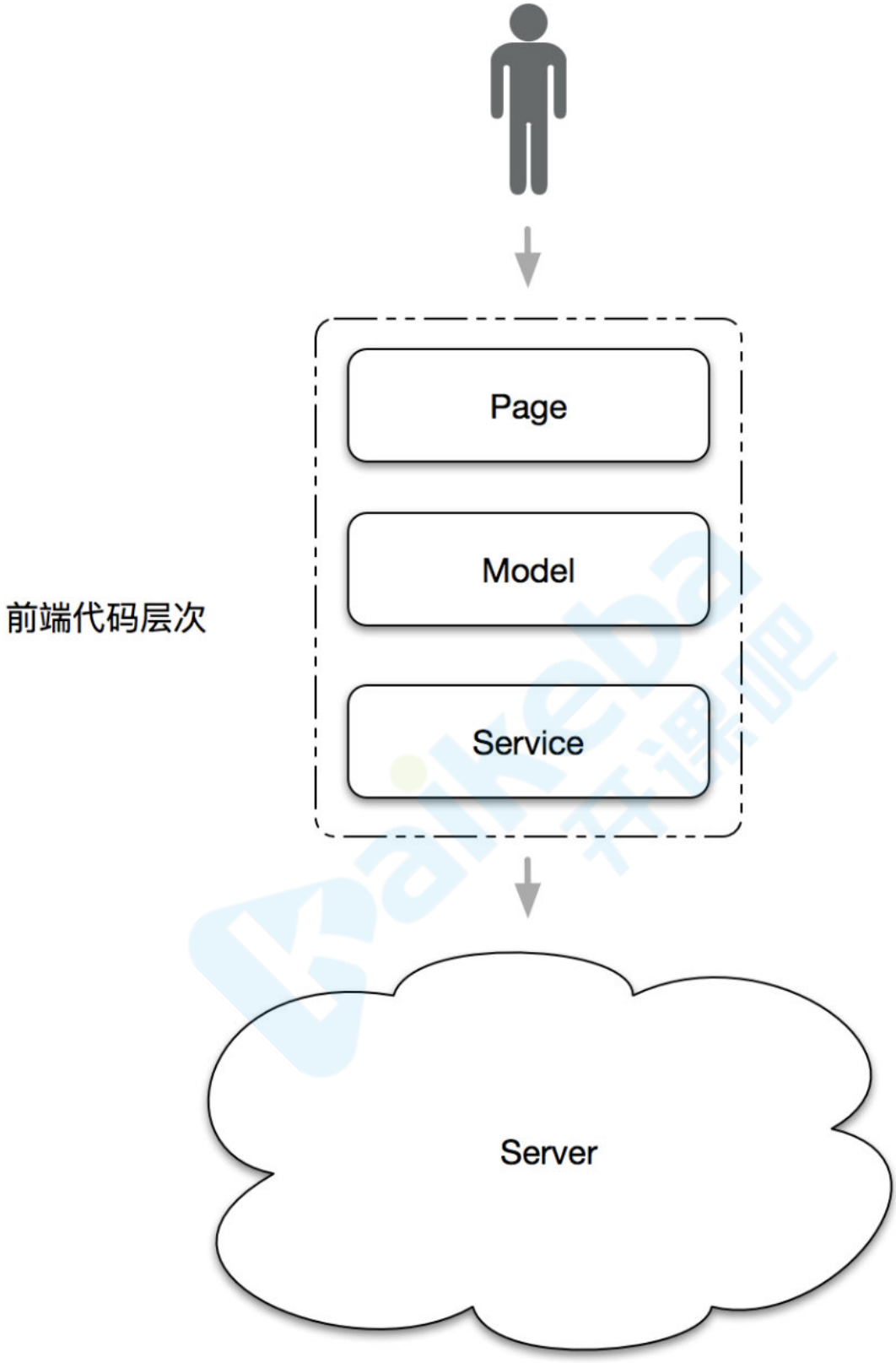
```
umi dev
```

> 访问index: http://localhost:8000/

> 访问about: http://localhost:8000/about

理解dva

软件分层：回顾react，为了让数据流更易于维护，我们分成了store，reducer，action等模块，各司其职，软件开发也是一样



1. Page 负责与用户直接打交道：渲染页面、接受用户的操作输入，侧重于`展示型交互性逻辑`。

2. Model 负责处理业务逻辑，为 Page 做数据、状态的读写、变换、暂存等。
3. Service 负责与 HTTP 接口对接，进行纯粹的数据读写。

DVA 是基于 redux、redux-saga 和 react-router 的轻量级前端框架及最佳实践沉淀，核心api如下：

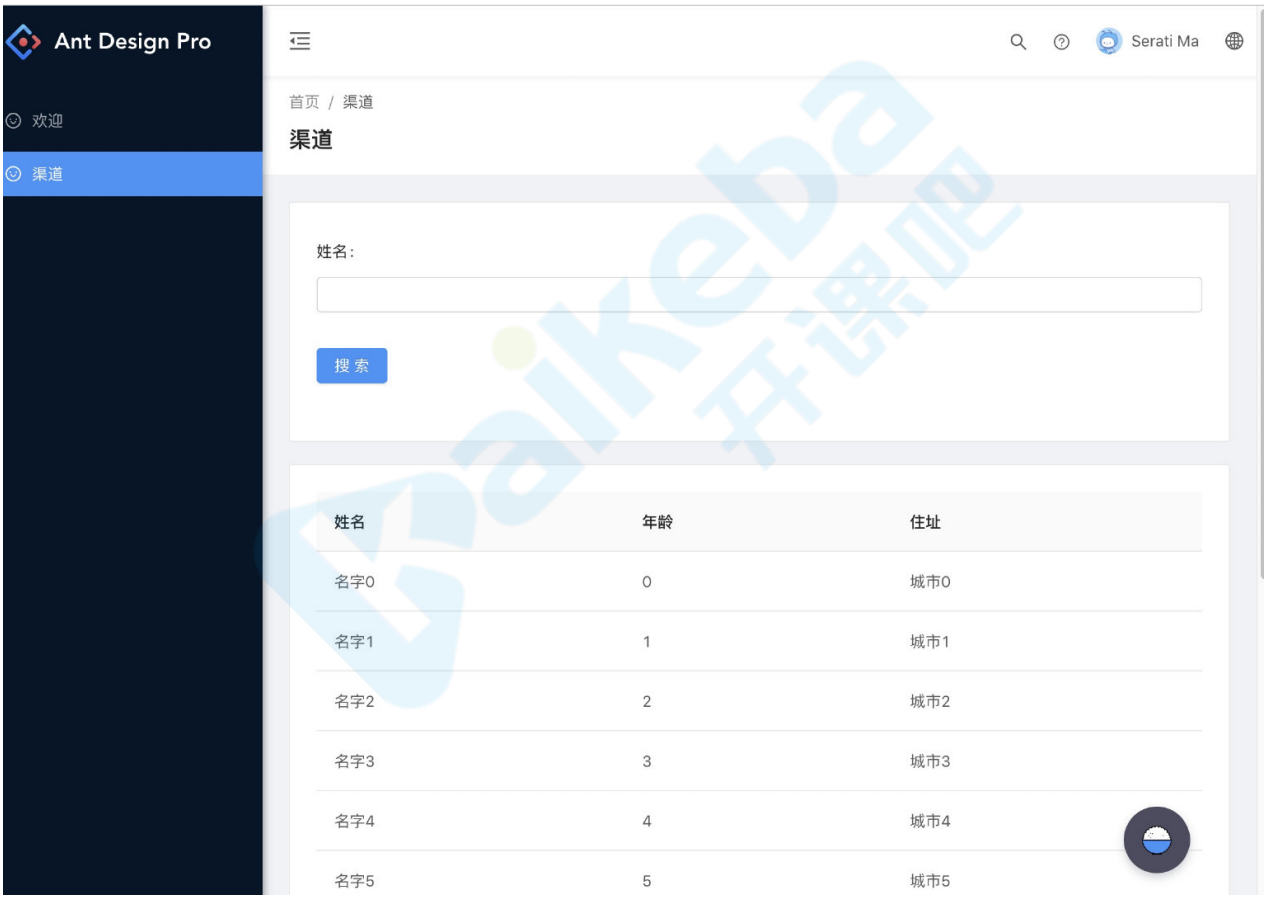
1. model
    - state 状态
    - action
    - dispatch
    - reducer
    - effect 副作用，处理异步
2. subscriptions 订阅
3. router 路由

> 1. `namespace`：model 的命名空间，只能用字符串。一个大型应用可能包含多个 model，通过`namespace`区分
> 2. `reducers`：用于修改 `state`，由 `action` 触发。reducer 是一个纯函数，它接受当前的 state 及一个 action 对象。action 对象里面可以包含数据体（payload）作为入参，需要返回一个新的 state。
> 3. `effects`：用于处理异步操作（例如：与服务端交互）和业务逻辑，也是由 action 触发。但是，它不可以修改 state，要通过触发 action 调用 reducer 实现对 state 的间接操作。

4. `action`：是 reducers 及 effects 的触发器，一般是一个对象，形如`{ type: 'add', payload: todo }`，通过 type 属性可以匹配到具体某个 reducer 或者 effect，payload 属性则是数据体，用于传送给 reducer 或 effect。

实例：

实现如下图：



使用状态：state + connect

- 创建页面channel.js：`umi g page channel/index --less`

```
import React, { Component } from 'react';
```

```
import styles from './index.less';
import { PageHeaderWrapper } from '@ant-
design/pro-layout';
import { Card, Form, Input, Button, Table }
from 'antd';
import { connect } from 'dva';

const columns = [
  {
    title: '姓名',
    dataIndex: 'name',
    key: 'name',
  },
  {
    title: '年龄',
    dataIndex: 'age',
    key: 'age',
  },
  {
    title: '住址',
    dataIndex: 'city',
    key: 'city',
  },
];

class Channel extends Component {
  componentDidMount() {
    this.props.getChannelData();
  }
```

```
  search = () => {
    const { getFieldValue } = this.props.form;
    const name = getFieldValue('name');
    this.props.getChannelDataBySearch({ name
});
  };

  render() {
    const { form, data } = this.props;
    const { getFieldDecorator } = form;
    return (
      <div className={styles.channel}>
        <PageHeaderWrapper>
          <Card className={styles.formCard}>
            <Form>
              <Form.Item label="姓名">
{getFieldDecorator('name')(<Input />)}
</Form.Item>
              <Form.Item label="城市">
{getFieldDecorator('city')(<Input />)}
</Form.Item>
            </Form>
            <Button type="primary" onClick=
{this.search}>
              提交
            </Button>
            <Button>重置</Button>
          </Card>
          <Card>
```

```
            <Table dataSource={data} columns=
{columns} rowKey={record => record.id} />
          </Card>
        </PageHeaderWrapper>
      </div>
    );
  }
}
export default connect(({ channel }) => ({
...channel }), {
  getChannelData: () => ({ type:
'channel/getChannelData' }),
  getChannelDataBySearch: search => ({ type:
'channel/getChannelDataBySearch', payload:
search }),
})(Form.create()(Channel));
```

- 更新模型src/models/channel.js

```
import { getChannelData, getChannelDataBySearch
} from '@/services/channel.js';

const model = {
  namespace: 'channel',
  state: {
    data: [],
  },
  effects: {
```

```
    *getChannelData({ payload }, { call, put })
{
      const response = yield
call(getChannelData, payload);
      yield put({
        type: 'channelData',
        payload: response,
      });
    },
    *getChannelDataBySearch({ payload }, {
call, put }) {
      const response = yield
call(getChannelDataBySearch, payload);
      console.log('has', response, payload);

      yield put({
        type: 'channelData',
        payload: response,
      });
    },
  },
  reducers: {
    channelData(state, { payload }) {
      return { ...state, data:
[...payload.data] };
    },
  },
};
```

```
export default model;
```

- 添加服务：src/service/channel.js

```
import request from '@/utils/request';
export async function getChannelData(params)
{
  return request('/api/getChannelData', {
    method: 'get',
  });
}
export async function
getChannelDataBySearch(params) {
  return
request('/api/getChannelDataBySearch', {
    method: 'post',
    data: params,
  });
}
```

数据mock：模拟数据接口

mock目录和src平级，新建mock/channel.js

```
const channelTableData = [];
for (let i = 0; i < 10; i++) {
  channelTableData.push({
    id: i,
    name: '名字' + i,
    age: i,
```

```
    city: '城市' + i,
  });
}
function searchChannelData(name) {
  const res = [];
  for (let i = 0; i < 10; i++) {
    if (channelTableData[i].name.indexOf(name)
> -1) {
      res.push(channelTableData[i]);
    }
  }
  return res;
}
export default {
  // 支持值为 Object 和 Array
  'GET /api/getChannelData': {//查询表单数据
    data: [...channelTableData],
  },
  'POST /api/getChannelDataBySearch': (req,
res) => {//搜索
    res.send({
      status: 'ok',
      data: searchChannelData(req.body.name),
    });
  },
};
```

移动端cra项目简介

所用技术： react、redux、react-redux、react-router-dom等等

项目安装： npm install

项目启动： npm start

mock:

cd mock-server

npm i

npm start

移动端适配

回顾

项目实战

- 课堂目标
- 资源
- 知识要点
- 起步
- Generator
- redux-saga
- umi
  - why umi
- 它和 dva、roadhog 是什么关系?

- dva
- dva+umi 的约定
- 安装
- Umi基本使用
- 理解dva

移动端cra项目简介

回顾