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. redux-saga：中文、英文
4. generator

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，详细参考参考阮一峰。

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使副作用（数据获取、浏览器缓存获取）易于管理、执行、测试和失败处理
- 地址：https://github.com/redux-saga/redux-saga
- 安装：**npm install --save redux-saga**
- 使用：用户登录

先创建一个RouterPage

```
import React, { Component } from "react";
import { BrowserRouter, Switch, Link, Route } from "react-
router-dom";
import LoginPage from "./LoginPage";
import UserPage from "./UserPage";
import PrivatePage from "./PrivatePage";

export default class RouterPage extends Component {
  render() {
    return (
      <div>
        <h1>RouterPage</h1>
        <BrowserRouter>
          <Link to="/login">登录</Link>
          <Link to="/user">用户中心</Link>
          <Switch>
            <Route path="/login" component={LoginPage} />
            {/* <Route path="/user" component={UserPage} />
*/}
```

```
            <PrivatePage path="/user" component={UserPage}
/>
          </Switch>
        </BrowserRouter>
      </div>
    );
  }
}
```

创建store/index.js

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

const initialLogin = {
  isLogin: false,
  loading: false,
  name: "",
  error: "",
};

function loginReducer(state = { ...initialLogin }, action)
{
  switch (action.type) {
    case "requestLogin":
      return {
        ...initialLogin,
        loading: true,
      };
    case "requestSuccess":
      return {
        ...state,
        isLogin: true,
        loading: false,
      };
    default:
```

```
      return state;
  }
}

const store = createStore(
  combineReducers({ user: loginReducer }),
  applyMiddleware(thunk),
);

export default store;
```

登录页面pages/LoginPage.js

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

import { connect } from "react-redux";

export default connect(
  //mapStateToProps
  state => ({
    isLogin: state.user.isLogin,
    loading: state.user.loading,
  }),
  {
    //mapDispathToProps
    /* login: () => ({
      type: "requestSuccess",
    }), */
    login: () => dispatch => {
      dispatch({ type: "requestLogin" });
      setTimeout(() => {
        dispatch({
          type: "requestSuccess",
        });
      }, 2000);
    },
```

```
  },
)(
  class LoginPage extends Component {
    render() {
      const { isLogin, loading, location, login } =
this.props;
      if (isLogin) {
        const { redirect = "/" } = location.state || {};
        return <Redirect to={redirect} />;
      }
      return (
        <div>
          <h1>LoginPage</h1>
          <button onClick={login}>{loading ? "登录中..." :
"登录"}</button>
        </div>
      );
    }
  },
);
```

路由守卫/pages/PrivatePage.js：

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

export default connect(
  //mapStateToProps
  state => ({
    isLogin: state.user.isLogin,
  }),
)(
  class PrivatePage extends Component {
    render() {
      const { path, component, isLogin } = this.props;
      if (isLogin) {
```

```
        return <Route path={path} component={component} />;
      }
      return (
        <Redirect
          to={{
            pathname: "/login",
            state: { redirect: path },
          }}
        />
      );
    }
  },
);
```

用saga的方式实现：

1. 创建一个./store/mySagas.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(name) {
    return new Promise((resolve, reject) => {
      console.log("omg");
      setTimeout(() => {
        if (name === "小明") {
          resolve({ name: "小明" });
        } else {
          reject("用户名或密码错误");
        }
      }, 1000);
```

```
    });
  },
};

//worker saga
function* loginHandle(action) {
  console.log("loginHandle");
  try {
    yield put({ type: "requestLogin" });
    const res = yield call(UserService.login, action.name);
    yield put({ type: "requestSuccess", res });
  } catch (err) {
    yield put({ type: "requestFailure", err });
  }
}

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

export default mySaga;
```

2. 创建user.redux.js，用户状态管理的reducer

```
const initialLogin = {
  isLogin: false,
  loading: false,
  name: "",
  error: "",
};

export default function loginReducer(state = {
...initialLogin }, action) {
  console.log("action", action);
  switch (action.type) {
    case "requestLogin":
```

```
      return {
        ...initialLogin,
        loading: true,
      };
    case "requestSuccess":
      return {
        ...state,
        isLogin: true,
        loading: false,
      };
    case "requestFailure":
      return {
        ...state,
        isLogin: false,
        loading: false,
        err: action.err,
      };
    default:
      return state;
  }
}
```

3. 注册redux-saga，./store/index.js

```
import { createStore, combineReducers, applyMiddleware }
from "redux";
// import thunk from "redux-thunk";
import createSagaMiddleware from "redux-saga";
import mySaga from "./mySaga";
import loginReducer from "./user.redux";

const sagaMiddleware = createSagaMiddleware();

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

```
);

sagaMiddleware.run(mySaga);

export default store;
```

4. 测试，LoginPage.js

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

import { connect } from "react-redux";

export default connect(
  //mapStateToProps
  state => ({
    isLogin: state.user.isLogin,
    loading: state.user.loading,
    err: state.user.err,
  }),
  {
    //mapDispathToProps
    /* login: () => ({
      type: "requestSuccess",
    }), */
    /* login: () => dispatch => {
      dispatch({ type: "requestLogin" });
      setTimeout(() => {
        dispatch({
          type: "requestSuccess",
        });
      }, 2000);
    }, */
    login: name => ({ type: "login", name }),
  },
)(
  class LoginPage extends Component {
```

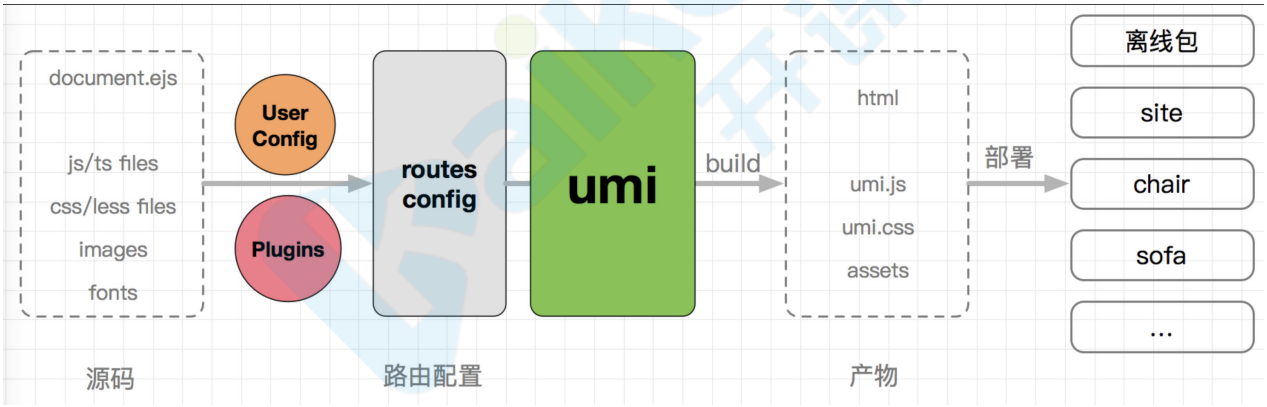
```
    constructor(props) {
      super(props);
      this.state = {
        name: "",
      };
    }
    setName = event => {
      this.setState({
        name: event.target.value,
      });
    };
    render() {
      const { isLogin, loading, err, location, login } =
this.props;
      console.log("pr", this.props);
      if (isLogin) {
        const { redirect = "/" } = location.state || {};
        return <Redirect to={redirect} />;
      }
      const { name } = this.state;
      return (
        <div>
          <h1>LoginPage</h1>
          {err && <p>{err}</p>}
          <input value={name} onChange={this.setName} />
          <button onClick={() => login(name)}>
            {loading ? "登录中..." : "登录"}
          </button>
        </div>
      );
    }
  },
);
```

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

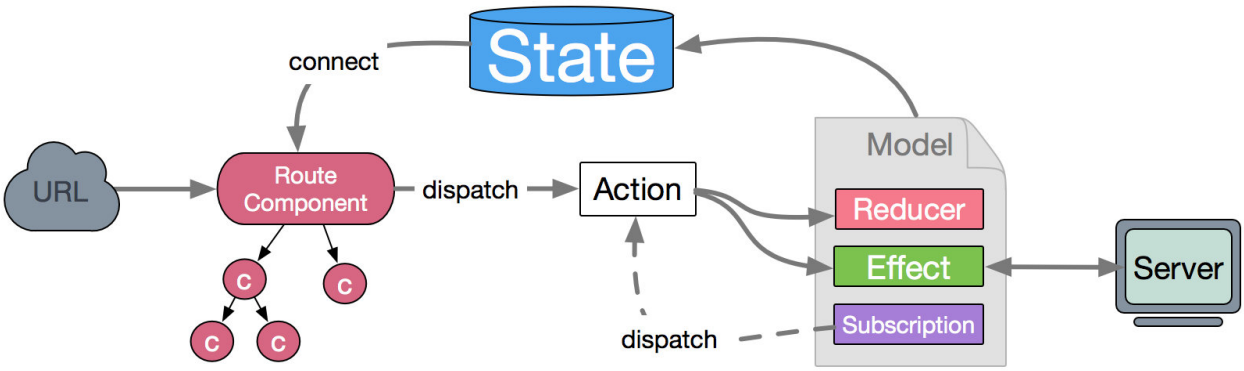
umi

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



dva



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 lesson6-1019-umi
进入文件夹：cd lesson6-1019-umi
创建：yarn create umi
选择ant-design-pro
选择Javascript
安装依赖： yarn 或者 npm install
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')

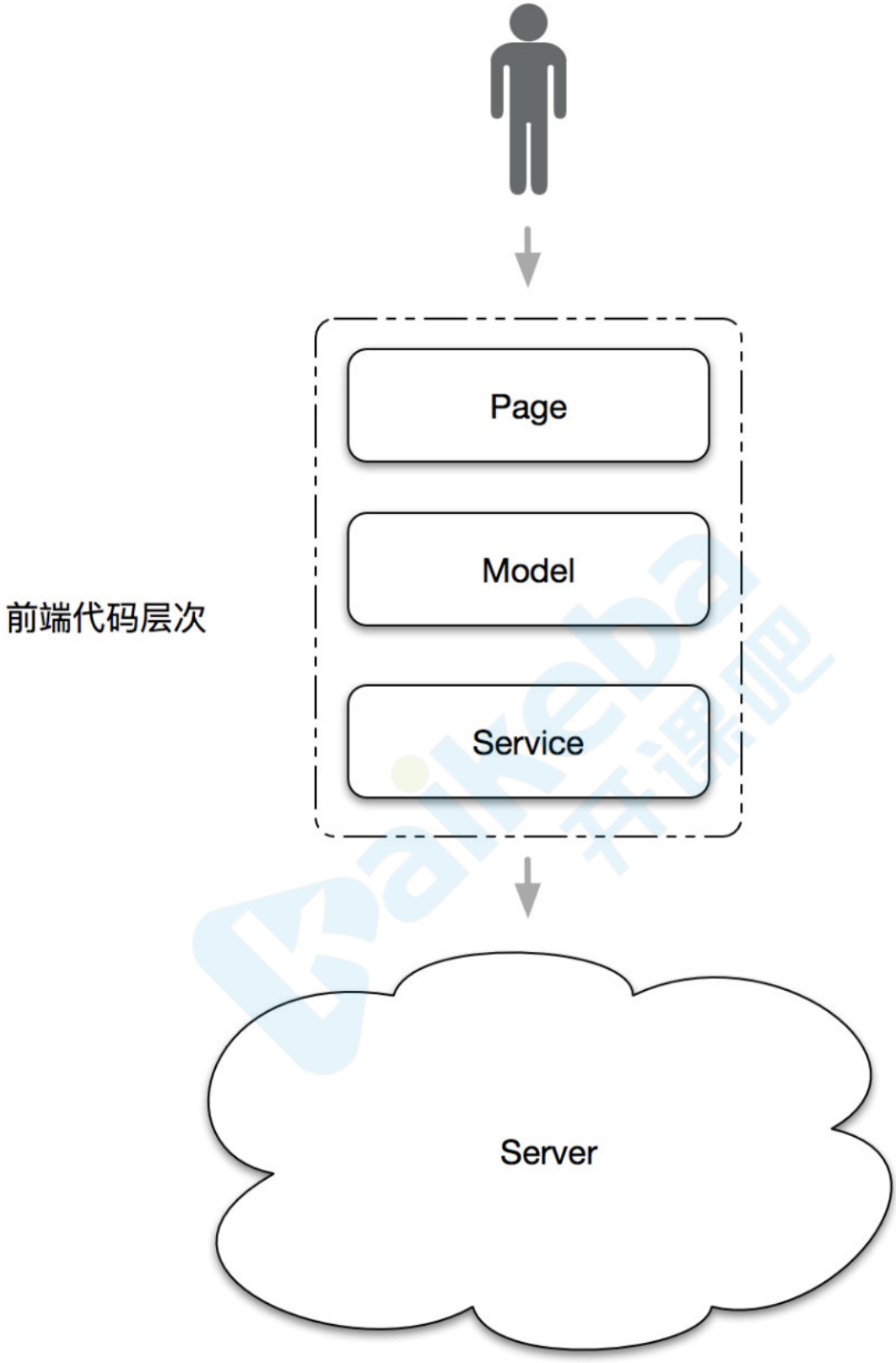
起服务看效果

```
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。

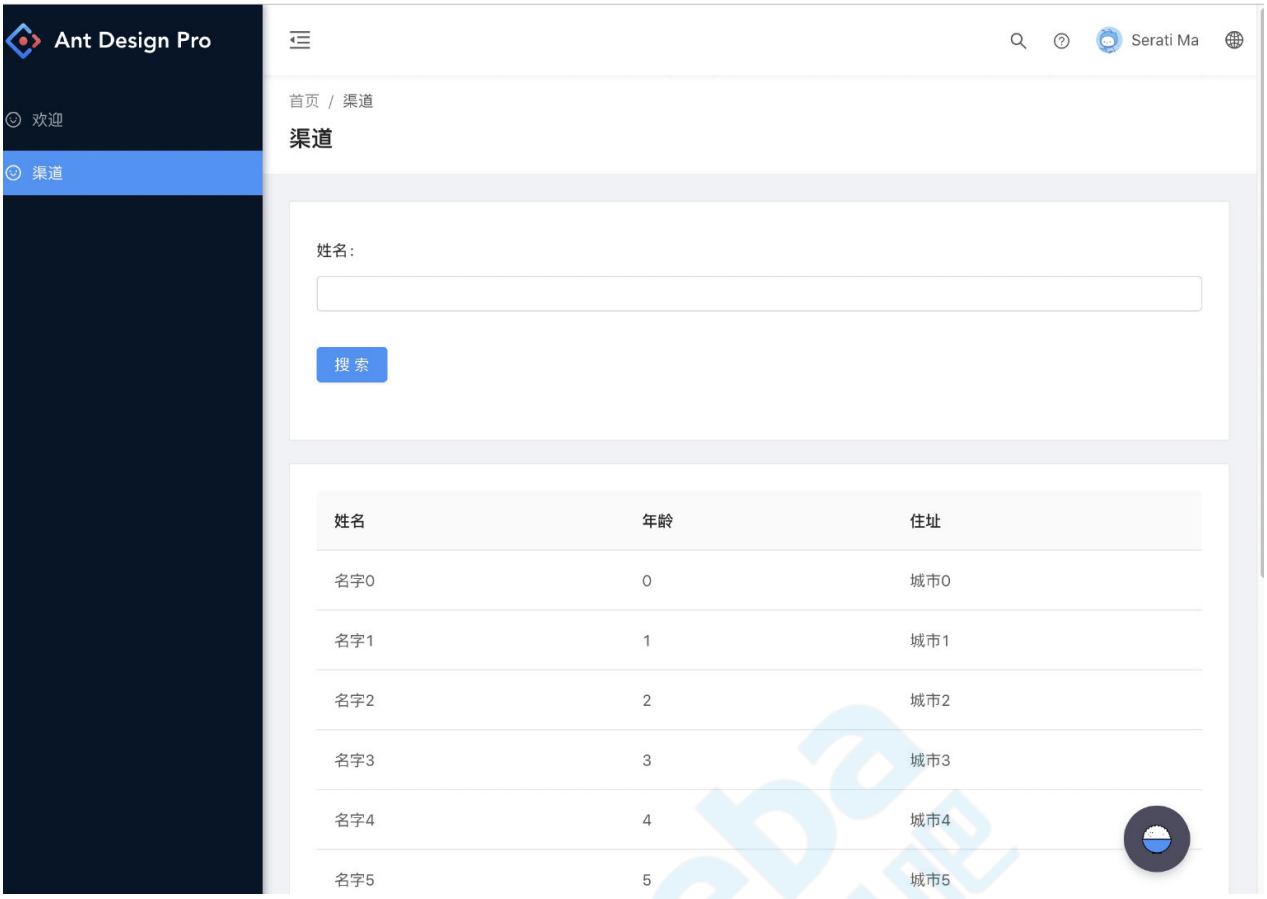
DVA 是基于 redux、redux-saga 和 react-router 的轻量级前端框架及最佳实践沉淀，核心api如下：

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goods.js：`umi g page channel/index --less`

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

const columns = [
  {
    title: '姓名',
    dataIndex: 'name',
    key: 'name',
  },
  {
    title: '年龄',
    dataIndex: 'age',
```

```
    key: 'age',
  },
  {
    title: '住址',
    dataIndex: 'city',
    key: 'city',
  },
];

export default connect(
  ({ channel }) => {
    return {
      ...channel,
    };
  },
  {
    getChannelData: () => {
      return { type: 'channel/getChannelData' };
    },
    getChannelDataBySearch: search => {
      return { type: 'channel/getChannelDataBySearch',
payload: search };
    },
  },
)(
  class Channel extends Component {
    constructor(props) {
      super(props);
      this.state = {
        name: '',
      };
    }
    componentDidMount() {
      this.props.getChannelData();
    }
    setFormValue = (name, event) => {
      this.setState({
```

```
        [name]: event.target.value,
      });
    };
    search = () => {
      const tem = { ...this.state };
      this.props.getChannelDataBySearch(tem);
    };
    render() {
      const { data } = this.props;
      const { name } = this.state;
      return (
        <div className={styles.channel}>
          <PageHeaderWrapper>
            <Card className={styles.formCard}>
              <Form>
                <Form.Item label="姓名">
                  <Input value={name} onChange={event =>
this.setFormValue('name', event)} />
                </Form.Item>
                <Form.Item>
                  <Button type="primary" onClick=
{this.search}>
                    搜索
                  </Button>
                </Form.Item>
              </Form>
            </Card>
            <Card>
              <Table dataSource={data} columns={columns}
rowKey={record => record.id} />
            </Card>
          </PageHeaderWrapper>
        </div>
      );
    }
  },
);
```

- 更新模型src/models/channel.js

```
import { routerRedux } from 'dva/router';
import { stringify } from 'querystring';
import { getChannelData, getChannelDataBySearch } from
'@/services/channel.js';
import { setAuthority } from '@/utils/authority';
import { getPageQuery } from '@/utils/utils';
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 }) {
      console.log('hah', payload);
      const response = yield call(getChannelDataBySearch,
payload);
      yield put({
        type: 'channelDataBySearch',
        payload: response,
      });
    },
  },
  reducers: {
    channelData(state, { payload }) {
      return { ...state, data: [...payload.data] };
    },
```

```
    channelDataBySearch(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