React最佳实践

# 资源

1. umi
2. dva
3. redux-saga
4. generator

# 起步

## redux-saga

安装：**npm install --save redux-saga**

使用：用户登录

创建一个./store/sagas.js处理用户登录请求

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

// 模拟登录
const UserService = {
  login(uname) {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        if (uname === "Jerry") {
          resolve({ id: 1, name: "Jerry", age: 18 });
        } else {
          reject("用户名或密码错误");
        }
      }, 1000);
    });
  }
};

// worker Saga
function* login(action) {
  try {
    yield put({ type: "requestLogin" });
    const result = yield call(UserService.login, action.uname);
    yield put({ type: "loginSuccess", result });
  } catch (message) {
    yield put({ type: "loginFailure", payload: message });
```

```
  }
}

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

export default mySaga;
```

2. 创建user.js，用户状态管理的reducer

```
export const user = (
  state = { isLogin: false, loading: false, error: "" },
  action
) => {
  switch (action.type) {
    case "requestLogin":
      return { isLogin: false, loading: true, error: "" };
    case "loginSuccess":
      return { isLogin: true, loading: false, error: "" };
    case "loginFailure":
      return { isLogin: false, loading: false, error: action.message };
    default:
      return state;
  }
};
// 派发动作依然是对象而非函数
export function login(uname) {
  return { type: "login", uname };
}
```

3. 注册redux-saga，./store/index.js

```
import { user } from "./user.redux";
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;
```

4. 测试，RouteSample.js

```
// Login
const Login = connect(
  state => ({
```

```
    isLogin: state.user.isLogin,
    loading: state.user.loading,
    error: state.user.error // 登录错误信息
  }),
  { login }
)(({ location, isLogin, login, loading, error }) => {// 登录错误信息
  const redirect = location.state.redirect || "/";

  // 若已登陆重定向至redirect
  if (isLogin) return <Redirect to={redirect} />;

  return (
    <div>
      <p>用户登录</p>
      <hr />
      {/* 显示错误信息 */}
      {error && <p>{error}</p>}
      {/* 登录传参 */}
      <button onClick={() => login('Jerry')} disabled={loading}>
        {loading ? "登录中..." : "登录"}
      </button>
    </div>
  );
});

// PrivateRoute
const PrivateRoute = connect(state => ({
  isLogin: state.user.isLogin
}))(function ({ component: Component, isLogin, ...rest }) {
    ...
}
```

# umi

## Umi基本使用

### 安装

```
npm install umi -g
```

### 项目目录

```
md umi-app
cd umi-app
```

### 新建 index页

```
生成首页
umi g page index
umi g page about
```

**起服务**

```
umi dev
```

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

**动态路由**

以$开头的文件或目录

```
umi g page users/$id
```

获取参数和以前写法相同

```
export default function({match}) {
  return (
    <div>
      <h1>user id: {match.params.id}</h1>
    </div>
  );
}
```

**嵌套路由**

目录下面出现_layout组件则会转换路由配置为嵌套路由

```
// 创建父组件 umi g layout ./users
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**

```
// 用户列表跳转至用户详情页, users/index.js
import Link from "umi/link";
import router from "umi/router";

export default function() {
  return (
    <div className={styles.normal}>
      <h1>用户列表</h1>
      <ul>
        {users.map(u => (
          // 声明式
```

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

#### 404页面

约定 `pages/404.js` 为 404 页面，需返回 React 组件。

创建404页面：`umi g page ./404`

#### 布局页

约定 `src/layouts/index.js` 为全局路由，返回一个 React 组件，通过 `props.children` 渲染子组件。

- 创建布局页面：`umi g page ../layouts/index`

```
export default function(props) {
  return (
    <div className={styles.normal}>
      <h1>布局页面</h1>
      {props.children}
    </div>
  );
}
```

- 针对特定路由指定布局页

```
if (props.location.pathname === '/404') {
    return <SimpleLayout>{ props.children }</SimpleLayout>
}
```

#### 通过注释扩展路由

约定路由文件的首个注释如果包含 **yaml** 格式的配置，则会被用于扩展路由。

权限路由，about.js

```
/**
 * title: About Page
 * Routes:
 *   - ./routes/PrivateRoute.js
 */
```

创建./routes/PrivateRoute.js

```
import Redirect from "umi/redirect";

export default props => {
  if (new Date().getDay() % 2 === 1) {
    // 单数日需要登陆
    return <Redirect to="/login"/>;
  }
  return (
    <div>
      <div>PrivateRoute</div>
      {props.children}
    </div>
  );
};
```

创建登录页面验证： `umi g page login`

## 引入dva

### 安装umi-plugin-react

```
npm install umi-plugin-react -D
```

### 配置

创建配置文件： .umirc.js

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

**创建model** 在src下建立
models

model用来维护页面数据状态

新建src/models/goods.js

```
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**

类似redux，使用connect获取数据状态并映射给组件

创建页面goods.js：`umi g page goods`

```
import { connect } from 'dva'
export default connect(
  state => ({
    goodsList: state.goods // 获取指定命名空间的模型状态
  }),
)(function({ goodsList }) {
  return (
    <div className={styles.normal}>
      <h1>Page goods</h1>
      <ul>
        {goodsList.map(good => (
          <li key={good.title}>{good.title}</li>
        ))}
      </ul>
    </div>
  );
});
```

更新模型src/models/goods.js

```
export default {
  reducers: {
    addGood(state, action) {
      return [...state, {title: action.payload}];
    }
  }
}
```

调用reducer：goods.js

```
export default connect(
  state => ({}),
  {
    addGood: title => ({
      type: "goods/addGood", // action的type需要以命名空间为前缀+reducer名称
      payload: title
```

```
    })
  }
)(function({ goodsList, addGood }) {
  return (
    <div className={styles.normal}>
      <button onClick={() => addGood("商品" + new Date().getTime())}>
        添加商品
      </button>
    </div>
  );
});
```

**数据mock**：模拟数据接口

mock目录和src平级，新建mock/goods.js

```
let data = [
  {title:"web全栈"},
  {title:"java架构师"}
];

export default {
  // "method url": Object 或 Array
  // "get /api/goods": { result: data },

  // "method url": (req, res) => {}
  'get /api/goods': function (req, res) {
    setTimeout(() => {
      res.json({ result: data })
    }, 250)
  },
}
```

**effect处理异步**：基于redux-saga，使用generator函数来控制异步流程

请求接口，models/goods.js

```
// 首先安装axios
import axios from 'axios';

// api
function getGoods(){
  return axios.get('/api/goods')
}

export default {
  state: [
    // {title:"web全栈"},
    // {title:"java架构师"},
```

```
    // {title:"百万年薪"}
  ],
  effects: { // 副作用操作，action-动作、参数等，saga-接口对象
    *getList(action, {call, put}){
      const res = yield call(getGoods)
      yield put({ type: 'initGoods', payload: res.data.result })
    }
  },
  reducers: {
    initGoods(state,{payload}){
      return payload
    }
  }
}
```

组件调用，goods.js

```
import {useEffect} from 'react';
export default connect(
  state => ({}),
  {
    addGood: title => ({}),
    getList: () => ({ // 获取数据
      type: "goods/getList"
    }),
  }
)(function({ goodsList, addGood,getList }) {
  useEffect(()=>{ // 调用一次
    getList();
  },[])
  return ();
});
```

**加载状态**

利用内置的dva-loading实现，获取加载状态，goods.js

```
connect(
  state => ({
    loading: state.loading // 通过loading命名空间获取加载状态
  }),
  {...}
)(function({ goodsList, addGood, getList, loading }) {
    console.log(this.props.loading);
    if (this.props.loading.models.goods) {
      return <div>加载中...</div>
    }
    ...
}
```

# 项目

## 引入antd

- 添加antd：`npm install antd -S`
- 修改.umirc.js

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

## 布局+导航

antd布局组件使用，修改layouts/index.js

```
import { Layout, Menu } from "antd";
import styles from "./index.css";
import Link from "umi/link";

const { Header, Footer, Content } = Layout;

export default function(props) {
  const selectedKeys = [props.location.pathname];
  return (
    <Layout>
      {/* 页头 */}
      <Header className={styles.header}>
        <img className={styles.logo} src="https://img.kaikeba.com/logo-new.png"/>
        <Menu
          theme="dark"
          mode="horizontal"
          selectedKeys={selectedKeys}
          style={{ lineHeight: "64px", float: 'left' }}
        >
          <Menu.Item key="/goods">
            <Link to="/goods">商品</Link>
          </Menu.Item>
          <Menu.Item key="/users">
            <Link to="/users">用户</Link>
          </Menu.Item>
          <Menu.Item key="/about">
            <Link to="/about">关于</Link>
          </Menu.Item>
        </Menu>
      </Header>
      {/* 内容 */}
```

```
        <Content className={styles.content}>
          <div className={styles.box}>{props.children}</div>
        </Content>
        {/* 页脚 */}
        <Footer className={styles.footer}>开课吧</Footer>
      </Layout>
    );
}
```

## 用户登录认证

引入ant-design-pro，安装：`npm install ant-design-pro --save`

测试：修改404页面提示内容，404.js

```
// umi的配置，已经自动支持antd-pro的按需加载
import {Exception} from 'ant-design-pro'
export default function() {
  return (
    <Exception type="404" backText="返回首页"></Exception>
  );
}
```

登录页构建，login.js：

```
import React, { Component } from "react";
// import { Button } from "antd";
import styles from "./login.css";
import router from "umi/router";
import { Login } from "ant-design-pro";

const { UserName, Password, Submit } = Login; // 通用的用户名、密码和提交组件

export default function(props) {
  let from = props.location.state.from || "/"; // 重定向地址
  const onSubmit = (err, values) => {
    console.log(err, values);
  };
  return (
    <div className={styles.loginForm}>
        {/* logo */}
        <img className={styles.logo}
          src="https://img.kaikeba.com/logo-new.png"/>
        {/* 登录表单 */}
        <Login onSubmit={onSubmit}>
          <UserName
            name="username"
            placeholder="kaikeba"
            rules={[{ required: true, message: "请输入用户名" }]}
          />
```

```
          <Password
            name="password"
            placeholder="123"
            rules={[{ required: true, message: "请输入密码" }]}
          />
          <Submit>登录</Submit>
        </Login>
      </div>
  );
}
```

登录接口mock，创建./mock/login.js

```
export default {
  "post /api/login"(req, res, next) {
    const { username, password } = req.body;
    console.log(username, password);
    if (username == "kaikeba" && password == "123") {
      return res.json({
        code: 0,
        data: {
          token: "kaikebaisgood",
          role: "admin",
          balance: 1000,
          username: "kaikeba"
        }
      });
    }
    if (username == "jerry" && password == "123") {
      return res.json({
        code: 0,
        data: {
          token: "kaikebaisgood",
          role: "user",
          balance: 100,
          username: "jerry"
        }
      });
    }
    return res.json({
      code: -1,
      msg: "密码错误"
    });
  }
};
```

用户信息保存和登录动作编写，创建./src/models/user.js

```
import axios from "axios";
import router from "umi/router";

// 初始状态：本地缓存或空值对象
```

```
const userinfo = JSON.parse(localStorage.getItem("userinfo")) || {
  token: "",
  role: "",
  username: "",
  balance: 0
};

// 登录请求方法
function login(payload) {
  return axios.post("/api/login", payload);
}

export default {
  namespace: "user", // 可省略
  state: userinfo,
  effects: {
    // action: user/login
    *login({ payload }, { call, put }) {
      const { data: {code, data: userinfo} } = yield call(login, payload);
      if (code == 0) {
        // 登录成功：缓存用户信息
        localStorage.setItem("userinfo", JSON.stringify(userinfo));
        yield put({ type: "init", payload: userinfo });
        router.push('/');
      } else {
        // 登录失败：弹出提示信息，可以通过响应拦截器实现
      }
    }
  },
  reducers: {
    init(state, action) {
      // 覆盖旧状态
      return action.payload;
    }
  }
};
```

请求登录，login.js

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

export default connect()(function(props) {
  onSubmit = (err, values) => {
    console.log("用户输入：", values);
    if (!err) {
      // 校验通过，提交登录
      props.dispatch({ type: "user/login", payload: values });
    }
  };
  ...
})
```

登录失败处理：

~ 设置响应状态码，./mock/login.js

```
// 设置401状态码
return res.status(401).json({
    code: -1,
    msg: "密码错误"
});
```

~ 响应拦截，创建./src/interceptor.js

```
import axios from "axios";
import { notification } from "antd";

const codeMessage = {
  202: "一个请求已经进入后台排队（异步任务）。",
  401: "用户没有权限（令牌、用户名、密码错误）。",
  404: "发出的请求针对的是不存在的记录，服务器没有进行操作。",
  500: "服务器发生错误，请检查服务器。"
};

// 仅拦截异常状态响应
axios.interceptors.response.use(null, ({ response }) => {
  if (codeMessage[response.status]) {
    notification.error({
      message: `请求错误 ${response.status}: ${response.config.url}`,
      description: codeMessage[response.status]
    });
  }
  return Promise.reject(err);
});
```

~执行拦截器设置代码，创建./src/global.js

```
// 全局入口
import interceptor from './interceptor'
```

~ saga中异常处理，修改./src/models/user.js

```
*login({ payload }, { call, put }) {
      try {
        // 同之前，删除else部分
      } catch (error) {
        // 登录失败：错误信息已在拦截器实现，可执行其他业务
      }
}
```

## 商品列表

数据mock，./mock/goods.js

图片素材，./public/courses/*.png

修改商品数据模型，pages\models\goods.js

```
export default {
  namespace: "goods",
  state: { // 初始状态包括课程和分类
    courses: {}, // 课程
    tags: [] // 分类
  },
  effects: {
    *getList(action, { call, put }) {
      // 解构出courseData并初始化状态
      const { data: { data: courseData } } = yield call(getGoods);
      yield put({ type: "initGoods", payload: courseData });
    }
  },
  reducers: {
    initGoods(state, { payload }) {
      // 解构出tags和courses并返回
      const { tags, data: courses } = payload;
      return { ...state, tags, courses };
    },
  }
};
```

显示课程分类页签，pages\goods.js

```
import { TagSelect } from "ant-design-pro";

@connect(
  state => ({
    courses: state.goods.courses, // 映射课程数据
    tags: state.goods.tags, // 映射标签数据
  }),
  {...}
)
class Goods extends Component {
  // 页签变更
  tagSelectChange = (tags) => {
    console.log(tags);
  };
  render() {
    if (this.props.loading.models.goods) {
      return <div>加载中...</div>;
    }
    return (
      <div>
        {/* 分类标签 */}
        <TagSelect onChange={this.tagSelectChange}>
          {this.props.tags.map(tag => {
```

```
            return (
              <TagSelect.Option key={tag} value={tag}>
                {tag}
              </TagSelect.Option>
            );
          })}
        </TagSelect>
      </div>
    );
  }
}
export default Goods;
```

显示课程列表，pages\goods.js

```
import { Card, Row, Col, Skeleton, Icon } from "antd";

class Goods extends Component {
  constructor(props) {
    super(props);
    // displayCourses为需要显示的商品数组
    this.state = {
      displayCourses: new Array(8).fill({}) // 填充数组用于骨架屏展示
    };
  }
  // 数据传入时执行一次tagSelectChange
  componentWillReceiveProps(props){
    if(props.tags.length){
      this.tagSelectChange(props.tags, props.courses)
    }
  }
  // 额外传入课程列表数据
  tagSelectChange = (tags, courses = this.props.courses) => {
    console.log(tags);
    // 过滤出要显示的数据
    let displayCourses = [];
    tags.forEach(tag => {
      displayCourses = [...displayCourses, ...courses[tag]];
    });
    this.setState({ displayCourses });
    console.log(displayCourses);
  };
  render() {
    // 使用骨架屏做加载反馈，loading属性不再需要
    // if (this.props.loading.models.goods) {
    //   return <div>加载中...</div>;
    // }
    return (
      <div>
        {/* 分类标签 */}
        {/* 商品列表 */}
        <Row type="flex" justify="start">
          {this.state.displayCourses.map((item, index) => {
```

```
            return (
              <Col key={index} style={{ padding: 10 }} span={6}>
                {item.name ? (
                  <Card
                    hoverable
                    title={item.name}
                    cover={<img src={"/course/" + item.img} />}
                  >
                    <Card.Meta
                      description={
                        <div>
                          <span>￥{item.price}</span>
                          <span style={{ float: "right" }}>
                            <Icon type="user" /> {item.solded}
                          </span>
                        </div>
                      }
                    />
                    <div />
                  </Card>
                ) : (
                  <Skeleton active={true} />
                )}
              </Col>
            );
          })}
        </Row>
      </div>
    );
  }
}
export default Goods;
```

TagSelect初始状态调整：默认应当全选

```
constructor(props) {
    super(props);
    this.state = {
      //...
      tags: [], // 默认未选中任何标签
    };
}

tagSelectChange = (tags, courses = this.props.courses) => {
    // 用户行为修改状态
    this.setState({ displayCourses, tags });
};

// 组件受控
<TagSelect value={this.state.tags}>
```

## 添加购物车

创建购物车模型，models/cart.js

```
export default {
  namespace: "cart", // 可省略
  state: JSON.parse(localStorage.getItem("cart")) || [], // 初始状态：缓存或空数组
  reducers: {
    addCart(cart, action) {
      const good = action.payload;
      const idx = cart.findIndex(v => v.id == good.id);
      if (idx > -1) {
        // 更新数量
        const cartCopy = [...cart];
        const itemCopy = { ...cartCopy[idx] };
        itemCopy.count += 1;
        cartCopy.splice(idx, 1, itemCopy);
        return cartCopy;
      } else {
        // 新增
        return [...cart, { ...good, count: 1 }];
      }
    }
  }
};
```

请求添加购物车，pages\goods.js

```
@connect(
  state => ({ ... }),
  {
    addCart: item => ({ // 加购方法
      type: "cart/addCart",
      payload: item
    }),
  }
)
class Goods extends Component {
  addCart = (e, item) => {
    e.stopPropagation();
    this.props.addCart(item);
  };
  render() {
      <Card extra={
          <Icon onClick={e => this.addCart(e, item)}
            type="shopping-cart"
            style={{ fontSize: 18 }} />}>
  }
}
```

购物车数据同步到localStorage

```
effects: {
    *addCart({payload}, {put, select}) {
        yield put({type: 'add', payload})

        const cart = yield select(state => state.cart);
        localStorage.setItem("cart", JSON.stringify(cart));
    }
  },
  reducers: {
    add(cart, action) {...}
  }
```

页头添加购物车信息，./src/layouts/index.js

```
import {Badge, Icon} from 'antd';

export default function(props) {
    render(){
        ...
        <div style={{float:'right'}}>
            <Icon type="shopping-cart" style={{fontSize:18}}/>
            <span>我的购物车</span>
            <Badge count={5} offset={[-4,-18]}/>
        </div>
    }
}
```

~ 希望徽章小一点，添加全局样式，./src/global.css

```
.ant-badge-count{
    height: 16px;
    border-radius: 8px;
    min-width: 16px;
    line-height: 16px;
    padding: 0;
}
```

~ 显示购物车数据

```
import { Dropdown } from "antd";

export default connect(state => ({ // 连接购物车状态
  count: state.cart.length,
  cart: state.cart
}))(function(props) {
    render(){
        // 构造购物车列表菜单
        const menu = (
          <Menu>
```

```
            {this.props.cart.map((item, index) => (
              <Menu.Item key={index}>
                {item.name}×{item.count} <span>￥{item.count * item.price}</span>
              </Menu.Item>
            ))}
          </Menu>
        );
        ...
        return (
          {/* 购物车信息放在Dropdown以便展示 */}
          <Dropdown overlay={menu} placement="bottomRight">
            <div className={styles.cart}>
              {/* 购物车项目数量 */}
              <Badge count={this.props.count} offset={[-4, -18]} />
            </div>
          </Dropdown>
        )
    }
})
```

## 作业

把你手头项目拿umi架构做一下练手

使用create umi脚手架工具创建项目并熟悉它展示的各种业务功能