

# 一种改进的 MVC 框架在 ERP 系统开发中的应用

叶小艳,张 芒,李伟民

(广州大学华软软件学院,广东 广州 510990)

**摘 要:** 现有的信息系统开发多数采用前后端分离架构技术来提高系统效率,但信息系统开发中 MVC 架构视图具有过度依赖模型、过程复杂以及性能不够优化的缺点。为了克服这些缺点,提出了基于浏览器页面渲染的改进前后端分离 MVC 模型。该模型前端采用 dva+React 框架,后端以快速开发平台的 WebAPI+React 为框架,弥补了原快速开发平台页面单调、响应式差的缺点。在该框架的基础上,实现了前后端分离的生态式信息原型系统。以 ERP 系统中库存、OA 等模块为例,搭建基于前后端分离的系统架构,以此搭建一个具有响应式布局的 SPA 后台 ERP 管理系统。使用 mocha 测试框架编写测试用例去测试 dva 框架与 React 框架,最终执行测试用例,系统未发现严重的缺陷,其业务逻辑无错误,其安全性无漏洞。该系统很好地满足了用户需求,开发周期短,大幅提高了开发效率。

**关键词:** ERP 系统; MVC; React 框架; 快速开发技术; 生态式

**中图分类号:** TP315

**文献标识码:** A

**文章编号:** 1673-629X(2018)09-0194-05

doi:10.3969/j.issn.1673-629X.2018.09.040

## Application of an Improved MVC Framework in ERP System Development

YE Xiao-yan, ZHANG Mang, LI Wei-min

(South China Institute of Software Engineering, Guangzhou 510990, China)

**Abstract:** The development of existing information system mainly adopts front-and-rear-end separation architecture to improve its efficiency, but the MVC architecture view in information system development has the disadvantages of over-dependence on model, complex process and insufficient optimization of performance. For this, we propose improved MVC model with front-and-rear-end separation based on the browser page rendering. The front end of the model adopts the dva+React framework and the rear-end adopts the WebAPI+React framework, to make up for the shortcomings of the tedious page and poor response of the fast-developing original platform. On the basis of the framework, we realize a ecological information prototype system for the separation of front and rear. Then taking storage and OA in ERP system as an example, we build a system architecture based on the front and back end, so as to construct a SPA background ERP management system with responsive layout. Mocha test framework is adopted to write test cases to test the framework of dva and React, with the test cases finally executed. The system is found no serious defects such as correct business logic and security free from loopholes. Therefore, it better meets the needs of users, shortens development cycle and greatly improves the efficiency of development.

**Key words:** ERP system; MVC; React framework; rapid development technology; ecological type

## 0 引言

在多终端的“互联网+”时代,如何提高系统开发效率是许多企业一直探讨的问题。经过多年的研究及尝试,前后端分离架构技术成为了许多企业的选择<sup>[1]</sup>。因为终端有 Android、IOS、Web,它们有不同的展现形式,但是需要展示的数据是一样的,所以只需要后台提供接口,前端(终端)获取数据并展示出来即可,并且

不同的终端需要不同的开发人员,只有前后端分离的架构最适合当前多终端的时代,因此前后端分离架构便成了趋势<sup>[2]</sup>。

传统开发协作模式有两种:一种是前端写静态页面,后端套模板<sup>[3]</sup>。静态页面可以进行本地开发,只需实现视图而不需考虑业务逻辑<sup>[4]</sup>。但因后端要套模板且浏览前端代码,过程较为复杂。另一种是前端写模

收稿日期:2017-10-09

修回日期:2018-02-26

网络出版时间:2018-05-16

**基金项目:** 广东大学生科技创新培育专项资金项目(pdjh2017b0997);广州大学华软软件学院院级质量工程重点建设专业项目(ZDZY201701);广州大学华软软件学院科研项目(ky201614)

**作者简介:** 叶小艳(1981-),女,硕士,讲师,研究方向为系统分析与快速开发技术;张 芒,硕士,高级工程师,研究方向为企业信息系统。

**网络出版地址:** <http://cnki.net/kcms/detail/61.1450.tp.20180515.1651.038.html>

板<sup>[5]</sup>,但前端编写过程中过于依赖后端环境,一旦后端没完成,前端则无法操作。这两种协作方式是目前市场上大量采用的“前后端分离”技术,但这两种方式都存在问题,且没有实现真正意义上的系统整体架构的前后端分离。

文中采用改进传统的 MVC 框架来实现前后端分离,以此减少前后端的耦合性,提高用户体验,加快网页加载速度,降低开发和维护成本,提升效率。

## 1 改进的 MVC 框架

### 1.1 快速开发技术及现有架构

传统的开发模式存在需求沟通不对称、需求变更频繁、需求不明确等问题,造成项目效率低下、延期甚至失败等问题。近年来逐渐兴起的快速开发技术,则是实现以业务为驱动的智能开发。形成的各种快速开发平台是基于业务导向的设计理念,抽提所有管理系统运行的驱动共性形成的“业务驱动模型”,省去了复杂和重复的编码过程,通过对智能报表、数据维护业务控制和其他参数的管理,可以快速、高效地开发各类业务系统。简化了系统的运行机制,抽提构成系统的稳定元素和个性元素,解决各类管理类软件的构筑元素,所以这种快速开发平台既可以适用于开发任何类型管理软件,又可以大幅度提高开发效率,减少技术瓶颈。

目前国内已有的 Web 快速开发框架有不少,仅在 MVC 方面,就有 struts 和 webwork,还有将 struts 和 webwork 统一的 struts2,以及 tapestry、JSF、easyJWeb<sup>[6]</sup>。权限管理框架有 Spring Security,异步调用技术有 AJAX、DWR,RIA 技术有 extjs、jQuery、FLEX、GWT 等<sup>[7]</sup>。现有的 Web 快速开发框架功能较强,但也有开发难度大和需要的知识复杂等缺点。这样开发人员不仅要熟悉服务器端语言,还必须掌握 Ajax 相关的难以调试的前端技术。与此同时,现有的 Web 快速开发框架从不同层面解决了开发过程中的部分问题,偏重某一方面,但仍然包含有相应的模板设计和枯燥的代码段。文中旨在整合并改进现有的 MVC 框架,并结合现有快速开发技术平台快速搭建业务模块的优点,构建一套生态式的开发方法。

MVC 模式诞生于 19 世纪 70 年代,流行至今。MVC 模式,即“模型—视图—控制器”的框架技术,是将一个应用的处理流程按照这种方式进行分离<sup>[8]</sup>。这样,一个应用流程体系分为模型、视图和控制器三个核心模块,分别在系统中承担不同的功能和责任。这种框架技术使开发更加高效,代码耦合度尽量减小,使应用程序各部分的职责更加清晰。

传统 MVC 模式如图 1 所示,其缺点如下:第一,视图依赖于模型。如没有模型,视图亦无法呈现效果;

第二,请求须经“控制器→模型→视图”固定流程,用户才可看到最终展现界面,过程过于复杂;第三,渲染视图的过程在服务端完成,呈现给浏览器的是带有模型的视图页面,性能无法更好地优化。

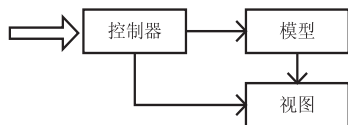


图 1 传统 MVC 模型

### 1.2 改进 MVC 架构

为了克服这些缺点,对模型进行改进。从浏览器发送 AJAX 请求到控制器,服务端接受请求,然后返回 JSON 数据给浏览器,直接在浏览器中渲染视图<sup>[9]</sup>,如图 2 所示。

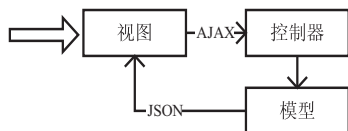


图 2 改进的 MVC 模型

将服务器那一端视为后端,浏览器这一端视为前端,将以上改进后的 MVC 模式简化为以下前后端分离模式。前端关注界面展现,后端关注业务逻辑,分工明确,职责清晰。

结合快速开发技术平台和改进后的 MVC 模型,构建系统开发架构<sup>[10]</sup>,如图 3 所示。

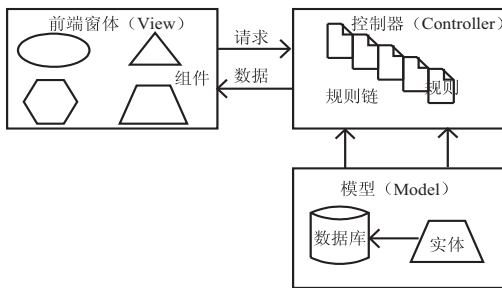


图 3 PC 系统开发架构

前端窗体通过各类组件组成,形成视图层;利用规则、函数等组成规则链来对视图层进行控制,并加入实体层,规则链可以对视图层与实体层进行操作控制,使用规则链来处理数据,使用实体模型来绑定窗体里的组件。调用数据时从数据库加载到实体,通过绑定实体的窗体呈现。

## 2 ERP 系统设计与实现

### 2.1 系统设计

#### 2.1.1 系统架构

系统以 ERP 系统中库存、OA 等模块为例,搭建基于前后端分离的系统架构,以此搭建一个具有响应式布局的 SPA 后台 ERP 管理系统。

基于图 3 开发架构的控制层与模型层,将视图层脱离出来,React+dva 的前端技术组建系统的前端页面,并通过访问控制层公开的接口对后台进行操作,最终形成了该系统架构,如图 4 所示(V3 指某快速开发平台)。

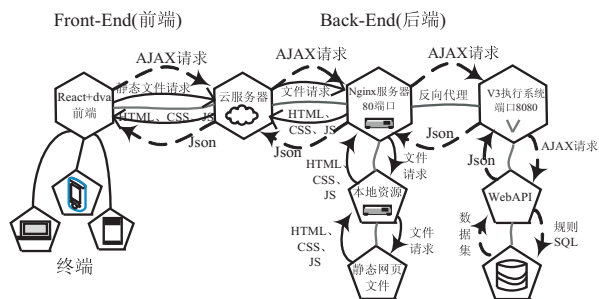


图 4 系统架构

系统分为前后端两部分：

(1) 前端向云服务器发起静态文件请求,如 HTML、CSS、JS、Image 等,由 Nginx 服务器搭建的 HTTP 服务会访问本地资源,得到 HTML、CSS、JS、Image 等文件,并返回到前端,然后展示通过浏览器编译成用户可以看到的页面。

(2) Nginx 通过反向代理将 V3 执行系统的地址代理到 80 端口,比如访问 <http://www.mitarl.com/api> 时,通过反向代理后最终访问的是 <http://www.mitarl.com:8080/webapi/>。

(3) 前端通过 AJAX (GET、POST) 向 Nginx 服务器发出请求,访问 V3 的 WebAPI 接口,WebAPI 通过规则链对数据进行处理,最后通过访问数据库并得到相应的数据,然后以 JSON 格式的数据返回到前端,前端拿到数据后对其进行处理,并展示在用户的浏览器上。

### 2.1.2 系统技术架构

系统技术架构如图 5 所示。

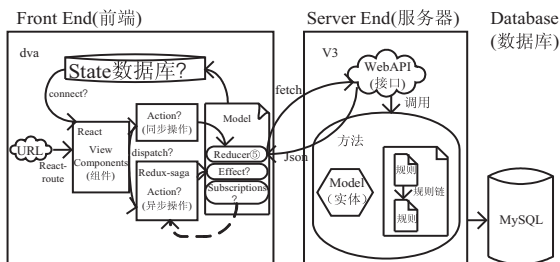


图 5 系统技术架构

①State 表示 Model 的状态数据；

②Action 是一个带 type 属性指明具体行为的对象,它是 View 层改变 State 的唯一途径,通过调用 dispatch 方法,传递 Action 到 Model 层,根据 Reducer 改变 State；

③异步的 Action,与同步的 Action 在 dispatch 时

是一样的,但是到达 Model 层,Model 层会先根据 type 先触发 Reducer 再触发 Effect；

④dispatch 是一个用于触发 Action 的函数,可以看作是触发这个行为的方式<sup>[11]</sup>；

⑤Reducer 是描述如何改变数据的<sup>[12]</sup>；

⑥Effect 被称为副作用,dva 底层引入 redux-saga 做异步流程控制<sup>[12]</sup>,运用 ES6 的 generator 的相关概念将异步转成同步写法；

⑦Subscription 订阅,然后根据条件 dispatch 需要的 Action,可以监听路由、键盘输入等的变化；

⑧connect 将 React 与 Redux 数据连通,通过映射 State 里面的数据,以 React 中 props 数据的方式传递给组件。

### 2.1.3 系统框架整合

先订阅监听浏览器的路由变化,当地址匹配成功后,会触发向服务器获取数据的异步请求,获取到数据后,对状态容器进行赋值,状态容器发生改变后映射到了对应的组件,组件就会进行渲染处理<sup>[13]</sup>。

(1)Subscriptions 订阅代码。

配置() { 监听路由 (路由地址) { 如果路由地址 = =/home 或者 =/= 则触发 getOwnerTasks 方法 } }

(2)Effect 代码(主要处理异步请求)。

getOwnersTask() { 先从 state 树里获取当前状态的 app 树下的 user 和 Home 树下的 query,再创建传递给接口的数据对象,然后调用写好的 service 接口并传入参数。等待 call 的结果,判断结果是否正确,正确就 put 触发对应的 Reducer 去修改状态树 }

(3)Service 服务接口代码。

一个异步函数,提供给 effect 调用,以 JSON 格式的数据传递 params 参数到服务端接口,最终收到接口的返回数据并传递给正在等待的 effect。

(4)Reducer 和 state 树。

Reducer 接收 action,即 put 时带的参数,这里是把接口传过来的参数 (todo、done 两个数组) 合并到当前的 state 树里。

(5)Connect 将 Redux 与 React 连接起来。

将 state 树中 Home 节点,app 节点下的 isNavbar、user 映射到 Home 组件,使得 Home 组件的 prors 属性存在这几个属性。

(6)Home 组件下的 Table 组件。

数据源是通过 state 树传过来 React 的 props 属性下的 todo,滚动条是根据浏览器大小决定的。使用 React 进行组件编写,通过 dva (Redux + React - Route + Redux-saga) 提供的 connect 组件对 React 和 Redux 进行通信,从而 React 组件可以获取 Redux 里面的 store 数据,React 通过发送 action 去触发 effect (异步) 请求

V3WebAPI 接口或 Reducer(同步),从而 Redux 改变 store 里面的数据,React 组件的 props 属性绑定了 Redux 的 store,store 改变 props 也会改变,React 组件就会自动刷新状态。

## 2.2 系统实现

基于篇幅,本节只介绍系统开发过程中用到的框架技术的典型示例。

### 2.2.1 登录功能

登录功能主要是对用户身份进行验证,核对用户输入的用户名、密码和验证码的数据合法性和一致性。登录功能采用异步请求到快速开发平台的登录构件,实现过程如下:

点击登录按钮会触发 action,然后 action 触发 effect(带\*的是 ES6 的异步函数),接收两个参数。第一个参数:接收 action 传过来的数据,这里是把这个参数里面的 payload 拿出来;第二个参数:获取 dva 框架提供的两种异步方法,call 是去请求 service(定义请求接口,类似于 Ajax)并传入 payload,通过 yield 关键字去等待异步数据,当异步数据获取回来后,赋值给 res, res 里面的 data 就是从 V3WebAPI 获取过来的数据,再判断数据是否正确,如果正确则 put 一个 Reducer,将传过来的数据对应修改 store 里面的数据,React 再刷新组件状态。

### 2.2.2 流程定义功能

快速开发平台支持完整流程定义和扩展,提供高度可视化流程设计器,集流程图设计、规则定制和代码扩展、调试于一体,流程设计开发快捷高效<sup>[14]</sup>。使用 iframe 连通快速开发平台的窗体,使快速开发平台的窗体能在自己的系统中使用,将自己写的页面与快速开发平台的窗体结合起来。

### 2.2.3 组织机构管理功能

组织机构管理也是使用 iframe 连接快速开发平台的窗体,因为快速开发平台已经提供了一套组织机构管理的标准的窗体与规则,完全不用自己去做任何的业务编写,所以利用快速开发平台的这些窗体来对系统进行扩展。

总之,使用 iframe 来连接快速开发平台提供的窗体,简化系统的一些必要业务,如 OA、组织机构管理等模块,可以直接引入快速开发平台的窗体来对系统进行扩展,从而节约开发时间和简化开发过程。

## 2.3 系统测试

### 2.3.1 Model 层测试

Model 层使用 Mocha 测试框架与第三方断言模块 expect 进行测试。Model 层测试代码如下:

```
import expect from 'expect';//引入断言模块
import {effects} from 'dva/saga';//引入 effects
```

```
import Home from '../src/models/Home';//引入需要测试的 model
```

```
//Home 测试用例
```

```
describe('Model 的 Home 测试用例',()=>{
  it('加载',()=>{expect(Home).toExist();});
  describe('测试 reducers',()=>{
    it('getOwnerTasksReducer',()=>{
      const reducers=Home.reducers;
      const reducer=reducers.getOwnerTaskReducer;
      const state={
        todo:[],
        done:[],
      };
      expect(reducer(state,
        {payload:{todo:[{id:1,state:'Running'}],done:[{id:2,
state:'Complete'}]}})
      )
      .toEqual({todo:[{id:1,state:'Runing'}],done:[{id:2,
state:'Complete'}]});
    });
  });
  describe('测试 effects',()=>{
    it('getOwnerTasks',()=>{
      const {call,put}=effects;
      const sagas=Home.effects;
      const saga=sagas.getOwnerTasks;
      const generator=saga({type:'. Home/getOwnerTasks'},
{call,put});
      let next=generator.next();
      //访问接口成功后,接口会返回是否成功的参数,正常情况
下是返回成功
      expect(next.value).toEqual({success:true});
    });
  });
});
```

Model 层需要测试的是 Reducer 与 Effect,Reducer 是普通函数,并且是纯函数,职责单一,对于固定输入,就有固定输出,所以很容易测试。Effect 的测试主要是验证这件事是否发起了对某个服务的调用,这个服务是否在执行,无关本模块的正确性。一个 Effect 实际上是转化为同步逻辑的测试,因为它是一个 generator 函数,只需对这个 Effect 一路 next,就能跑完整个逻辑。

### 2.3.2 表现层测试

表现层主要是 React 组件,利用 Enzyme 库结合 mocha 与 expect 断言,去测试 React 组件的虚拟 DOM 创建情况、虚拟 DOM 的渲染情况等。表现层测试代码如下:



```
import React from 'react';
import { shallow, mount, render } from 'enzyme';
import Home from '../src/routes/Home';
import MinimzeCard from '../src/components/common/
MinimzeCard';

describe('Home 测试虚拟 DOM', function() {
  it('Home 下的第一个 MinimzeCard 的 title 属性是待办',
function() {
    let app=shallow(<Home/>);
    //在 Home 的虚拟 DOM 中寻找 MinimzeCard 组件,找到它的
    title 属性,它的断言为待办
    expect(app.find(MinimzeCard).at(@).props('titled')).to.
equal(待办);
  });
});

describe('Home 测试渲染情况', function() {
  it('渲染 Home 组件,渲染出两个 MinimzeCard', function()
{
    let app=render(<Home/>);
    //渲染 Home 组件后,寻找 MinimzeCard 组件的长度大小,
    断言为 2
    expect(app.find(MinimzeCard).length).to.equal(2);
  });
});
```

总体而言,使用 mocha 测试框架编写测试用例去测试 dva 框架与 React 框架,最终执行测试用例,并通过所有的测试用例,最后系统未发现严重的缺陷,系统业务逻辑无错误,保证了系统的完整无漏洞。

### 3 结束语

该系统 PC 端,使用某公司快速开发平台,实现以业务为驱动的智能开发,更多考虑用户的需求,实现复杂的企业级管理系统软件的开发,极大地提升了软件的开发效率和开发质量,大幅缩短了开发周期<sup>[15]</sup>。

系统移动端采用前后端分离技术以及响应式布局,并使用流行前端框架 React 开发了 ERP 系统,经过详细的设计和测试,最终实现了整个开发框架。React 的组件化思想,使得系统更加模块化,更容易维护、拓展。用户可以较快地熟悉系统,可以利用 PC 端和移动端操作系统,从而提高工作效率,缩短开发流程。

利用快速开发技术,以业务为驱动快速开发系统,将重点放在系统更完善的功能上而非代码,结合现有移动端的开发框架,逐步完善移动端框架,并且对移动

端扩展更多的功能。推广这种快速、扩展性强的框架方法,使信息系统开发更加便捷多元。

### 参考文献:

- [1] ABOUTALEBI M, MOTAMENI H, AKBARPOUR N. A new approach to evaluate correctness in graph based business process modeling languages (case study: BPMN) [C]//International conference on computer technology and science. [s. l.]:[s. n.],2012:135-139.
- [2] AKBARPOUR N, ABOUTALEBI M, AHMADIKATOULI A. SBPMN+:a new approach to business process modeling [C]//6th IEEE joint international information technology and artificial intelligence conferences. Chongqing, China: IEEE,2011.
- [3] CHEN Wenlan. Enterprise information management system based on J2EE and MVC mode[M]. Berlin:Springer,2014.
- [4] 黎吾鑫,王 新.基于 Extjs+Spring MVC 的 Web 系统框架及应用研究[J]. 云南大学学报:自然科学版,2013,35(S2):110-115.
- [5] 张 宇,王映辉,张翔南.基于 Spring 的 MVC 框架设计与实现[J]. 计算机工程,2010,36(4):59-62.
- [6] 刘红霞,陆文迪.改进的 MVC 设计模式的研究与应用[J]. 计算机工程与科学,2015,37(9):1688-1691.
- [7] 刘 亮,霍剑青,郭玉刚,等.基于 MVC 的通用型模式的设计与实现[J]. 中国科学技术大学学报,2010,40(6):635-639.
- [8] 全 茵.基于 ASP.NET MVC 模式的软件开发架构的研究与探讨[J]. 中国电子科学研究院学报,2016,11(6):599-602.
- [9] 陈 辉,丁春莉,孙 悦.ASP.NET MVC 软件架构模式在学生实训管理系统的应用[J]. 电子设计工程,2015,23(13):11-14.
- [10] 高 宏,王婷婷.基于 MVC 和 Entity Framework 的团餐系统[J]. 计算机与现代化,2014(12):64-68.
- [11] 刘耀钦,袁承芬.MVC 设计模式在 Web 开中的应用与研究[J]. 信息安全与技术,2013,4(11):78-80.
- [12] 赵 伟,王志华,周 兵.基于.NET 技术和 MVC 的新架构模式[J]. 计算机工程与设计,2012,33(7):2646-2651.
- [13] 司 颀.MVC 模式下的考试系统建模研究[J]. 计算机科学,2013,40(6A):403-406.
- [14] 任广震,侯 进,王 献.MVC 模式在 B/S 结构政务系统的应用研究[J]. 计算机应用与软件,2014,31(8):54-58.
- [15] 尼俊红,张 丽,张 森,等.基于 Ajax 和 MVC 的电力通信告警系统的设计实现[J]. 计算机应用与软件,2013,30(8):226-227.