锐竞采购平台

**前端架构说明文档**

**V2.0.0**

**版 本 历 史**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **作者** | **参与者** | **日期** | **描述** |
| 0.0.1 | 林宗圣 | 林宗圣 | 2016/12/28 | 背景需求，原理，架构图，伪代码 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 背景需求

锐竞采购平台，专注于科研试剂耗材采购。目的是为医院、院校、科研机构提供试剂耗材采购服务，着力解决试剂耗材采购过程中的信息不对称、渠道单一、价格虚高、监管困难、审计繁琐等问题。目前平台已经进驻多家医院，上千家供应商，上架百万种商品。平台月流水已超过千万，年流水过亿。平台还在快速发展期，进驻医院数量再逐步增加中。、

平台主要系统为：商品采购系统、商品竞价系统、医院管理系统、供应商管理系统

（1）商品采购系统

商品采购系统不仅具备商品展示，挑选商品、搜索商品、购物车、生成订单、收货、退货等常规的商城功能，还具备医院特色的经费卡、采购审批等功能。

（2）商品竞价系统

商品竞价系统，支持一次竞价采购多种商品，支持多家供应商报价，比对商品价格和供应商服务承诺后择优采购。

（3）医院管理系统

医院管理系统，具备丰富的管理功能，包括部门管理、人员管理、审批设置、供应商进驻管理、各种统计报表汇总等。

（4）供应商管理系

供应商管理系统，为供应商提供了资质申请、品牌申请、商品上下架、订单维护、结算等丰富的运营功能，减轻供应商经营压力，降低市场营销成本，从而提高供应商的竞争力。

# 技术选型及原理

前端使用的技术框架为backbone+jquery+require+underscore

（1）backbone.js

backbone.js已经不是当前最流行的前端框架了，但是依然具有比较好的学习价值。虽然目前来说，react，vue等mvvm框架非常火热，但是感觉自身还不到去使用这种框架的层次。这些技术虽好，但是对个人的挑战也是比较大：首先是在编程基础这个部分，包括数据结构，算法，面向对象编程，设计模式，设计原则等等；其次是工作方法层面，对比angualr,react以及vue，跟传统的用原生js或者jquery写的代码，包括html,css和js三个方面，你会发现这两种技术直接导致的我们在工作结果上的巨大差别，这对于已经习惯了传统开发的我来说，要挑战的不仅仅是新技术的学习跟研究，更多的是工作思路，工作方法甚至是跟其它同事配合协作方式的改变，这个难度也很大。再加上我本人是一个还比较喜欢去钻研细节的人，所以从开始做前端开发到现在，一直都没有大胆地学习很多的新东西，相反，我把更多的精力花在了编程思维的锻炼以及基础知识的巩固上，这件事情看起来很小，但是对技术的提高很有帮助。

我为什么在这段时间会去琢磨一个现在不是很火热的前端框架，而且还认为它有比较好的学习价值。因为基于backbone的Model与View的开发方式，或者说它提供的面向对象的代码组织方式，跟我目前惯用的思路还是比较像的，尽管我已有的代码都是jquery搞的。但是它又要优于我现在的编程方式，因为它里面有一个数据驱动UI的思想在里面，而且还有一个很好的内置的事件管理机制，使得它在一些封装层面的东西，比我写的更要严谨，清晰一点。只要是能够提高代码质量的东西，我都认为是编程基础的一部分，这正是我目前仍然想花时间去钻研的东西，所以我想学习。即使不用它做任何的项目，只要把它的思想，能够渗透到我现在的思维中即可，这个我感觉也不太容易，所以我得花一小段时间，才能掌握好它里面的一些机制。我学习它的主要方法是阅读官方文档和写东西实践，阅读文档过程中对于自己有疑问的api，必须写一些简单的代码才能知道它的详细作用。好在它官方文档组织地还不错，所以在学习过程中，需要去测试的api并不是很多。然后为了了解如何在实际工作中运用backbone编写代码以及它与我现在的编程方式上的区别，我分别用jquery跟backbone写了一个todo app。简单起见，jquery实现的版本与官网的功能完全一致，当然代码是不同的；backbone实现的版本在官网的基础上，考虑了异步回调处理以及操作的交互还有批量请求的处理，使得这个简单的app看起来更符合实际的产品需求。最后我发现，虽然这两个版本实现方式不同，但是思路层面却有相似性，这个去看一下两份代码中定义的类名就清楚了。

1. underscore

[underscore](http://github.com/jashkenas/underscore/) 是一个 JavaScript 工具库，它提供了一整套函数式编程的实用功能，但是没有扩展任何 JavaScript 内置对象。 他解决了这个问题：“如果我面对一个空白的 HTML 页面，并希望立即开始工作，我需要什么？” 他弥补了 [jQuery](http://jquery.com/) 没有实现的功能，同时又是[Backbone](http://backbonejs.org/) 必不可少的部分。

underscore 提供了100多个函数，包括常用的：map、filter、invoke — 当然还有更多专业的辅助函数，如：函数绑定、JavaScript 模板功能、创建快速索引、强类型相等测试等等。

1. require.js

require.js的目标是代码的模块化，它使用了不同于传统<script>标签的脚本加载步骤。可以用它来加速、优化代码，但其主要目的还是为了代码的模块化。它鼓励在使用脚本时以module ID替代URL地址。

requireJS以一个相对于[baseUrl](http://makingmobile.org/docs/tools/requirejs-api-zh/" \l "config-baseUrl)的地址来加载所有的代码。页面顶层<script>标签含有一个特殊的属性data-main，require.js使用它来启动脚本加载过程，而baseUrl一般设置到与该属性相一致的目录。

<script data-main=”scripts/main.js” src=”scripts/require.js”></script>

baseUrl亦可通过[RequireJS config](http://makingmobile.org/docs/tools/requirejs-api-zh/" \l "config)手动设置。如果没有显式指定config及data-main，则默认的baseUrl为包含RequireJS的那个HTML页面的所属目录。

RequireJS默认假定所有的依赖资源都是js脚本，因此无需在module ID上再加".js"后缀，RequireJS在进行module ID到path的解析时会自动补上后缀。你可以通过[paths config](http://makingmobile.org/docs/tools/requirejs-api-zh/" \l "config-paths)设置一组脚本，这些有助于我们在使用脚本时码更少的字。

有时候你想避开"baseUrl + paths"的解析过程，而是直接指定加载某一个目录下的脚本。此时可以这样做：如果一个module ID符合下述规则之一，其ID解析会避开常规的"baseUrl + paths"配置，而是直接将其加载为一个相对于当前HTML文档的脚本。

一般来说，最好还是使用baseUrl及"paths" config去设置module ID。它会给你带来额外的灵活性，如便于脚本的重命名、重定位等。同时，为了避免凌乱的配置，最好不要使用多级嵌套的目录层次来组织代码，而是要么将所有的脚本都放置到baseUrl中，要么分置为项目库/第三方库的一个扁平结构。

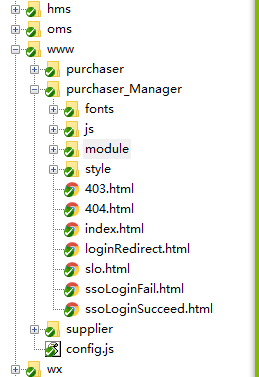
模块不同于传统的脚本文件，它良好地定义了一个作用域来避免全局名称空间污染。它可以显式地列出其依赖关系，并以函数(定义此模块的那个函数)参数的形式将这些依赖进行注入，而无需引用全局变量。RequireJS的模块是模块模式的一个扩展，其好处是无需全局地引用其他模块。

RequireJS的模块语法允许它尽快地加载多个模块，虽然加载的顺序不定，但依赖的顺序最终是正确的。同时因为无需创建全局变量，甚至可以做到在同一个页面上同时加载同一模块的不同版本。

# 架构图

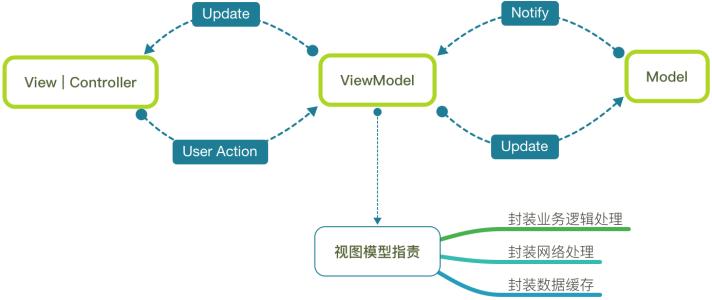
## 项目目录结构

单页面请求，整个框架只有一个入口，都从index.html进入，路径通过router的配置。（目前只采用backbone的router）



文件夹purchaser为采购平台门户，purchaser\_Manager为采购人中心，supplier为供应商， fonts保存字体，js封装了公共的模块，例如路由与某些公用函数比如ajax请求等，module为各功能模块的封装，style保存样式文件。

## 3.2 项目架构图



# 4.伪代码

## 4.1 路由js/router.js

var Router = Backbone.Router.extend({

routes: routesMap,

defaultAction: function() {

location.hash = '';

}

});

var router = new Router();

//彻底用on route接管路由的逻辑，这里route是路由对应的value

router.on('route', function(route, params) {

require([route], function(controller) {

controller.apply(null, params); //每个模块约定都返回controller

});

});

## 4.2锚点跳转 js/url.js

define(function() {

return {

allUrls: {

getSystemNotice:'/api/system/notice/findSystemNotice' // 首页系统公告

getProductInfo: '/product/getProductInfo', //商品详情信息

getShoppingCartList: '/user/getShoppingCartList', //购物车详情

getCartList:'/api/user/shoppingCart/LoadShoppingCart', //购物车列表

getProductList: '/api/product/findProduct', //搜索商品/列表

getProductCategory: '/api/product/category/findCategory', //商品分类

getFindPassPicCode: '/api/system/GetCaptcha', //图形验证码

getRootlist:"/column/helplist",//帮助中心栏目列表

getonlinesbycolumn:"/notice/getonlinesbycolumn",//帮助中心二级栏目列表

noticeDetails:"/notice/detail/"//帮助中心详情

}

};

});

## 4.3公用函数封装 js/common.js

define(['jquery', 'text!module/header/navigation.html'], function($, navHtml) {

var LOGIN\_IFRAME\_CLASS = "\_\_sso\_iframe\_\_";

// 提供服务

return {

pageNo: 1,

//分页条数

pageSize: 20,

//服务器地址（跨域）

serverBaseUrl: "/store",

providerBaseUrl: "/supp",

bidBaseUrl: "/bid",

cmsBaseUrl: "/cms",

getLoginIframe: function(ssoTarget) {

// 找到第一个正在处理登录请求的iframe

var iframes = $("iframe." + LOGIN\_IFRAME\_CLASS);

for(var i = 0, max = iframes.size(); i < max; ++i) {

var iframe = $(iframes[i]);

var done = iframe.prop('done');

var ssoTargetOfIFrame = iframe.prop('ssoTarget');

if(!done && ssoTargetOfIFrame == ssoTarget) {

return iframe;

}

}

return null;

},

fireRetriesInIframe: function(iframeId) {

var iframe = $(document.getElementById(iframeId));

if(iframe) {

var requests = iframe.prop('requests');

$.each(requests, function(idx, request) {

if(request[0]) {

request[0]();

}

});

iframe.prop('done', true);

iframe.remove();

}

},

fireCallbacksInIframe: function(iframeId) {

var iframe = $(document.getElementById(iframeId));

if(iframe) {

var requests = iframe.prop('requests');

$.each(requests, function(idx, request) {

if(request[1]) {

request[1]();

}

});

iframe.prop('done', true);

iframe.remove();

}

},

checkIfLoginRequired: function(response, url) {

// 获取不同应用中要求“请先登录”的错误码

var app = url.split("/");

if(app.length == 0) {

// 随便返回的。只要别让错误码判断错误就行

return "?";

}

app = app[1];

if(app == 'store') {

return response.code == "1020";

} else if(app == 'supp') {

return response.code == "1020";

} else if(app == 'bid') {

return response.resultCode == "1001";

} else if(app == 'cms') {

return response.resultCode == "1001";

}

// 随便返回的。

return "?";

},

loginInto: function(ssoLoginPath, successCallback, failCallback) {

// 简单的处理方式是，检查当前页面内是否已经存在特定类的iframe，如果已经存在，将请求放入登录后

// 应回调的处理队列中，然后跳过登录处理

var loginIframe = this.getLoginIframe(ssoLoginPath);

var requests;

if(loginIframe) {

requests = loginIframe.prop('requests');

var callbacks = [successCallback, failCallback];

requests.push(callbacks);

return;

}

// 登录使用checkOnly，即静默登录，遇到完全未登录的情况直接跳回cb\_fail\_target，静默

// 登录成功的情况跳转到cb\_target

var origin = window.location.origin;

var path = window.location.pathname.split("/").slice(0, -1).join("/");

var prefix = origin + path;

var cbTarget = prefix + "/ssoLoginSucceed.html";

var cbFailTarget = prefix + "/ssoLoginFail.html";

var ssoTarget = ssoLoginPath;

ssoLoginPath += "?cb\_target=" + encodeURIComponent(cbTarget) + "&cb\_fail\_target=" + encodeURIComponent(cbFailTarget) + "&checkOnly=true";

// cb\_target 在成功的时候回调。回调时，将requests中记录的所有请求重新发送一次（fireRetriesInIframe），finalTry字段

// 标记为true以避免再次发生需要登录的情况导致死循环。

// cb\_fail\_target 在无法静默登录时调用requests中记录的回调（fireCallbacksInIframe）。另一种

// 可以考虑的处理方式是直接不调用回调而跳到登录页面上。

// 在iframe处理完所有请求后把iframe移除。登录处理完成。

loginIframe = $("<iframe>");

loginIframe.prop("ssoTarget", ssoTarget);

loginIframe.prop('requests', []);

requests = loginIframe.prop('requests');

var iframeId = "login-" + Math.random();

loginIframe.prop("id", iframeId);

loginIframe.prop("class", LOGIN\_IFRAME\_CLASS);

loginIframe.css("display", "none");

requests.push([successCallback, failCallback]);

loginIframe.appendTo($(document.body));

loginIframe.prop("src", ssoLoginPath);

},

retryIfNotLoggedInOtherwiseCallback: function(response, url, params, callback, isJson, retryPredicate, method) {

if(retryPredicate) {

// 使用重试断言进行判断

if(!retryPredicate(response, url, params)) {

callback(response);

return;

}

} else {

// 重试断言未提供，使用默认的重试判断

// 返回码1020是请先登录的错误码。非1020的话直接调用回调。

// 但返回码的含义好像没统一……bid上是3000

if(!this.checkIfLoginRequired(response, url)) {

callback(response);

return;

}

}

// 1020 的情况下，尝试到CAS服务器登录，然后再重新发送一次请求

// 登录依旧走iframe登录，但需要注意postData可能在不同时刻被调用多次而iframe登录流程仍未完成，

// 所以需要注意判断iframe登录是否已经开始

// 构造CAS登录路径。对于接口的调用，形式是类似：

// http://host:port/store/api\_name

// 那么在这里的CAS登录路径就比较简单了，用第一段path+cas登录专用路径即可

var ssoLoginPath = url.split("/").slice(0, 2).join("/");

ssoLoginPath += "/sso/login";

method = method || "post";

var \_this = this;

if(method == "post") {

this.loginInto(ssoLoginPath,

function() {

\_this.postData(url, params, callback, isJson, false);

},

function() {

callback(response);

});

} else {

this.loginInto(ssoLoginPath,

function() {

\_this.getData(url, params, callback, isJson, false);

},

function() {

callback(response);

});

}

},

/\*\*

\* 请求方法

\* @param {string} url 请求的url

\* @param {json} params 传入的参数

\* @param {function} callBack 回调方法

\* @param {Boolean}isJson 是否为json格式

\* @param {Boolean}doRetryWhenRequireLogin 是否在“请先登录”的情况下先尝试登录然后再发送请求

\* @param {function} retryPredicate 是否需要登录后再重试一次请求的断言方法。如果不传入，默认从接口返回代码判断

\* \*\*/

postData: function(url, params, callBack, isJson, doRetryWhenRequireLogin, retryPredicate) {

var retry = (doRetryWhenRequireLogin == null || doRetryWhenRequireLogin == undefined) ? true : doRetryWhenRequireLogin;

var that = this;

var postData;

//兼容ie7（未解决）

if(isJson == true) {

if(window.JSON) {

postData = JSON.stringify(params);

} else {

postData = (new Function("return " + params))();

}

} else {

postData = params;

}

$.ajax({

url: url,

data: postData,

type: 'post',

dataType: 'json',

contentType: isJson == true ? 'application/json' : 'application/x-www-form-urlencoded',

/\* timeout: 20000, \*/

beforeSend: function() {

$('body').append("<span class='loading-gif' id='loading-gif'><img src='images/loading.gif' /></span>");

},

success: function(r) {

if(retry) {

that.retryIfNotLoggedInOtherwiseCallback(r, url, params, callBack, isJson, retryPredicate);

} else {

callBack(r);

}

},

complete: function() {

$(".loading-gif").remove();

}

/\* error: function(XMLHttpRequest, textStatus, errorThrown) {

if(XMLHttpRequest.statusText == "timeout") {

$.cookie('id', '', {

path: '/'

});

$.cookie('username', '', {

path: '/'

});

window.location.href = baseUrl + urls.allUrls.outLogin;

}

} \*/

});

}

/\*\*

\* 获取url中的字段

\* @params {name} string 获取字段的名字

\* return {context} 返回获取的字段

\* \*\*/

getQueryString: function(name) {

if(!/\?/.test(location.hash)) {

return false;

}

var \_reg = new RegExp("(^|&)" + name + "=([^&]\*)(&|$)", "i"),

\_hash = location.hash.split('?')[1],

\_r = \_hash.match(\_reg);

var \_context = "";

if(\_r != null) \_context = \_r[2];

\_reg = null;

\_r = null;

return \_context == null || \_context == "" || \_context == "undefined" ? "" : \_context;

},

/\*\*

\* 页码

\* @params {totalNum} 总条数

\* @params {pageIndex} 初始为第几页，默认0为第1页

\* @params {pageSize} 每页显示的条数

\* @params {pageCallback} 点击页数后执行的方法

\* \*\*/

pageFun: function(totalNum, pageIndex, pageSize, pageCallback) {

$("#pagination").pagination(totalNum, {

prev\_text: " ",

next\_text: " ",

current\_page: pageIndex - 1,

ellipse\_text: "...",

num\_edge\_entries: 2,

num\_display\_entries: 5,

items\_per\_page: pageSize,

callback: pageCallback,

link\_to: "javascript:void(0);"

});

}

};

});