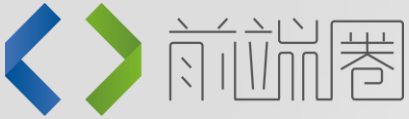


基于node.js的单页面开发

By 张敏聪

关于我

姓名：张敏聪

昵称：聪少

zhangmc@corp.21cn.com

21CN有史以来
身高最高前端

HTML5 node.js 前端自动化 用户体验 全栈 填坑小王子

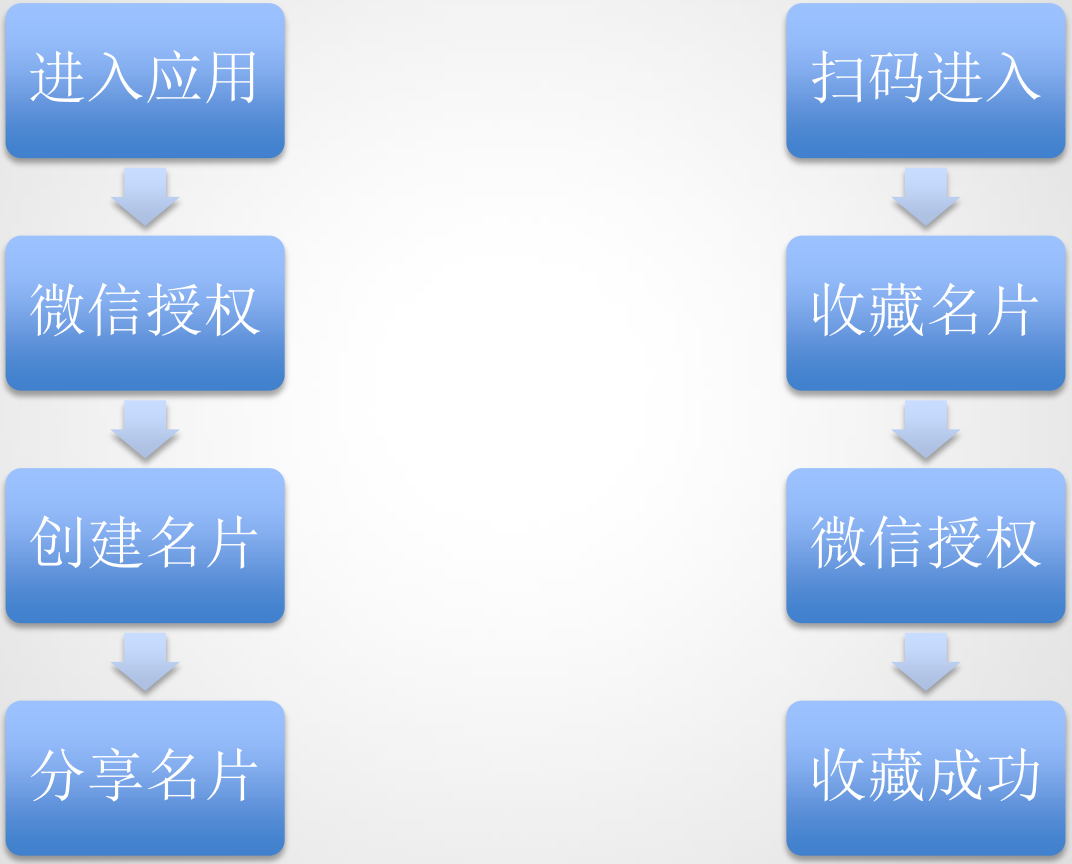
从开发一个单页面应用说起

要做什么

- 基于微信的wap名片
- 提供新增、分享、收藏、管理功能
- 名片采用vcard 3.0标准，要求能导出保存至手机通讯录

基本的交互流程

交互流程



功能与页面的划分

功能与页面的划分—我的


●●○○○ 中国移动 19:35 49%
返回 关闭
微-名片
张敏聪
21cn
我有185cm！！！
我的名片
创建新名片
我的收藏


●●○○○ 中国移动 19:36 49%
返回 关闭
微-名片
取消
创建新名片
保存
姓氏
请输入姓氏
名字
请输入名字
公司
请输入公司
添加电话
添加电子邮件
添加URL
我的名片
创建新名片
我的收藏


●●○○○ 中国移动 19:35 49%
返回 关闭
微-名片
王尼玛
暴走漫画
我是隔壁热心的老王~~
我的名片
创建新名片
我的收藏

功能与页面的划分—分享预览与下载

移动端单页面应用开发注意事项

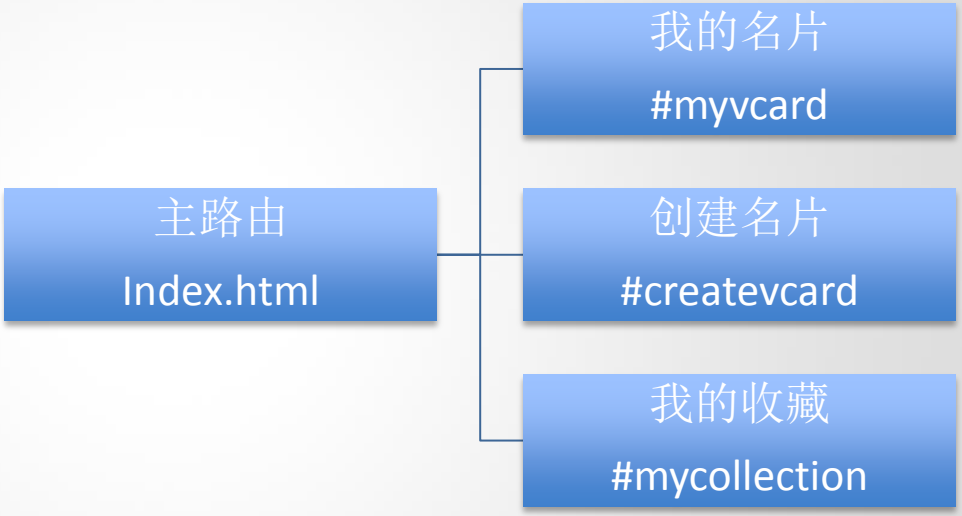
单页面应开发注意事项

- 页面性能与加载速度，按需加载
- 代码模块化，复用性，前后端通用
- 后端路由控制，JSON数据or页面直出
- 服务端框架选择——koa

前端模块化与后端路由中间件解读

前端js代码模块化

“我的”页面前端功能模块划分



Spm3前端模块组织与构建

- 遵循CommonJS规范
- spm-sea构建成CMD规范
- 配合seajs实现按需加载
- 使用seajs.alias配置，方便版本控制

commonJS规范编码与加载

```
var main;
var Backbone=require('backbone');
main=Backbone.Router.extend({
    routes:{
        createvcard:'createVcard',
        'myvcard':'myVcard',
        'mycollection':'myCollection',
        '*actions':'myVcard'
    },
    views:{},
    createVcard:function(){...},
    myVcard:function(){...},
    myCollection:function(){...},
    changeActiveView:function(newView){...}
});
window.appRouter=new main();
Backbone.history.start();
module.exports = main;
```

```
seajs.config({
    base:'/js/libs/',
    alias : {
        backbone:'backbone/1.1.2/back
        backbonevm:'backbone.vm/1.2.0
        jquery:'jquery/1.10.1/jquery.
        underscore:'underscore/1.6.0/
        vcard:'vcard/1.0.0/vcard.js',
        zepto:'zepto/1.1.3/index.js',
        iscroll:'iscroll/4.2.5/iscrol
        iscrolldatepicker:'iscrolldat
        'arale-qrcode':'arale-qrcode/
        //应用相关模块
        appmain:'plus/main/1.0.0/inde
        createvcard:'plus/createvcard
        myvcard:'plus/myvcard/1.0.0/m
        mycollection:'plus/mycollecti
        viewvcard:'plus/viewvcard/1.0
        share:'plus/share/1.0.0/share
    }
});
```

CommonJS规范编码实现代码前后端共用

- 名片数据采用vcard 3.0标准进行传输存储
- 创建时需要将输入数据转换成vcard
- 预览时需要将vcard转换成json再进行模渲染
- 遵循CommonJS规范，前后端共用一套逻辑

后端路由中间件使用

后端路由中间件使用

```
1  var index=require('koa-router')();
2  var config=require('../config/config')();
3  var wx_oauth2=require('../wx_oauth2/wx_oauth2');
4  var wxjsapi_ticket=require('./wxjsapi_ticket');
5  index.use(wx_oauth2);
6  index.get('/',wxjsapi_ticket,function *(next){
7      var wxjsapi_ticket=this.jsapi_ticket;
8      yield this.render('index',{wxjsapi_ticket:wxjsapi_ticket});
9  });
10
11 module.exports=index;
```

微信授权认证的接入流程

获取微信开发资源

- 测试号申请
- 获取测试号信息
- 配置授权回调域名
- 配置JS接口安全域名（若需要使用JS-SDK）

微信网页授权流程

- 引导用户进入授权页面同意授权，获取code
- 通过code换取网页授权access_token
- 刷新access_token（如果需要），保存access_token
- 通过网页授权access_token和openid获取用户基本信息

留白，故意的

一些后话

前后端分离后，前端目前的开发模式

前端的一些境况

- 更专注于UI、交互与体验，面向用户
- 以接口形式与服务端进行数据交互
- 偏离服务端，重度依赖于后台接口
- 整体开发能力的缺失

前后端开发的强助力——node.js

node.js

- 对前端友好，使用技术门槛低
- 一套语言，前后端通用
- 基于V8引擎，无浏览器兼容性等问题羁绊
- 活跃的生态圈支持

新一代的web开发框架——koa

koa

- 级联代码，既保持js异步编程特性，又保持代码可读性
- 基于中间件的处理形式，免除重复繁琐的回调函数嵌套
- 更高效的错误处理
- 拥抱ES6，拥抱新技术

koa VS express

完成JS-SDK签名生成—express版

```
'use strict';
var request=require('request');
var sign = require('sign');
var appId='wx5494a45a703e6541';
var appSecret='2501bf4cd0212afe45372a71597a9cb3';
var fnGetJsapi_ticket=function(req,res){
    var getAccessTokenUrl='getAccessTokenUrl'+'&appid='+appId+'&secret='+appSecret;
    request.get(getAccessTokenUrl,function (error, response, body) {
        var tokenResBody=JSON.parse(body);
        var getJsapi_ticketUrl='getJsapi_ticketUrl'+'?access_token='
            +tokenResBody.access_token+'&type=jsapi';
        request.get(getJsapi_ticketUrl,function(error, response, body){
            var ticket=JSON.parse(body);
            var url=req.protocol+'://'+req.host+req.url;
            var jsapi_ticket=sign(ticket, url);
            jsapi_ticket.appId=appId;
            res.json(jsapi_ticket);
        });
    });
};
module.exports=fnGetJsapi_ticket;
```

完成JS-SDK签名生成—koa版

```
'use strict';
var request=require('koa-request');
var sign=require('sign');
var appId='wx5494a45a703e6541';
var appSecret='2501bf4cd0212afe45372a71597a9cb3';
var getJSApiTicket=function *(){
    //get ticket from wx server
    var tokenUrl='tokenUrl'+'&appid='+appId+'&secret='+appSecret;
    var tokenResponse=yield request({url:tokenUrl});
    var tokenResBody=JSON.parse(tokenResponse.body);
    var getJsapi_ticketUrl='getJsapi_ticketUrl'+'?access_token='
        +tokenResBody.access_token+'&type=jsapi';
    var ticketResponse=yield request({url:getJsapi_ticketUrl});
    return JSON.parse(ticketResponse.body)
};
var wxjsapi_ticket=function *(next){
    var ticket=yield getJSApiTicket;
    var req=this.request;
    var url=req.protocol+'://'+req.host+req.url;
    var jsapi_ticket=sign(ticket, url);
    jsapi_ticket.appId=appId;
    this.state.jsapi_ticket=jsapi_ticket;
    yield next;
};

module.exports=wxjsapi_ticket;
```

生成JS-SDK签名的一些坑

JS-SDK

- jsapi_ticket有效期为7200秒，调用次数有限，必须服务端全局缓存
- 签名用的url必须是调用JS接口页面的完整URL，但不包括‘#’hash后面的部分；
- 签名用的nonceStr和timestamp必须与wx.config中的nonceStr和timestamp相同
- 官方签名生成node示例中，jssha版本注意使用1.5.0

谢谢