Lizard 2.0 接入手册

# 概论

Lizard2.0 是前端框架组为统一各PD业务代码以及SEO代码所推出的Lizard版本。Lizard2.0 废弃之前的Hash路由机制（在WEB环境下把路由交给服务端处理，使线上h5站点可具备A/B Testing能力；在Hybrid环境下，构建本地路由表模拟WEB下的路由能力），并采用多页面方式，有效解决之前单页应用对爬虫不友好的问题。

本文介绍Lizard2.0 框架基本原理，以及老版本1.1代码如何往2.0迁移。

# SEO方案介绍

## Lizard1.1/1.0路由机制

在1.1/1.0系统中，每个 View都是一个AMD模块，这些模块返回各自View构造函数。系统采用Hash即ViewId的机制，框架解析URL地址中的Hash,如果没有Hash则读取系统配置的默认ViewId。系统识别出ViewId后加载对应的 View模块，在加载的回调中，获得模块的返回（View的构造函数），创建View对象。

## Lizard2.0路由机制

Lizard 2.0项目本身是一个.net MVC项目，从某种意义上说已经不是一个纯前端项目。下面是Lizard2.0项目中的global.asax代码片段

|  |
| --- |
| public static void RegisterRoutes(RouteCollection routes)  {  routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");  routes.MapRoute(  "Default", // Route name  "{controller}/{action}/{id1}/{id2}/{id3}/{id4}", // URL with parameters  new {  controller = "Common",  action = "Index",  id1 = UrlParameter.Optional,  id2 = UrlParameter.Optional,  id3 = UrlParameter.Optional,  id4 = UrlParameter.Optional  } // Parameter defaults  );  }} |

浏览器输入URL地址，.net服务端根据global.asax中定义的路由规则到对应的目录下找到cshtml文件，编译成Html字符串作为response返回给浏览器，完成路由的功能。

# 代码架构介绍

Lizard2.0是一个.net MVC项目，其目录结构满足.net MVC项目目录结构。见下图：



其中web.config文件定义.net项目可以访问的全局变量，通常是一些路径相关的变量。Controllers文件夹是对应的控制类文件，让每个action都返回一个.net View对象.Views目录存放动态页面文件。

# Lizard2.0之旅

## 单页/多页

在前面介绍路由机制的时候已经提及了2.0 相比于1.1的差异。原先的1.1版本，只需编写一个静态HTML页面（通常是index.html）在这个页面中，引入公共资源（CSS），以及meta定义，框架JS，并有少量JS完成requireconfig配置。

Lizard2.0 将这些公共的配置抽取到header以及footer中，其中header引用CSS资源以及浏览器meta配置。footer引入2.0种子JS。另外种子SCRIPT标签中支持pdconfig属性，允许各个业务团队引入自己的requireconfig。

参见如下代码

|  |
| --- |
| <script type="text/javascript" src="@ConfigurationManager.AppSettings["WebresourceBaseUrl"]code/lizard/2.0/webapporigin/lizard.seed.js" pdConfig="@ConfigurationManager.AppSettings["WebresourcePDBaseUrl"]tour/tourConfig.js"></script> |

下面是一个页面的代码

|  |
| --- |
| @using System.Configuration  <!DOCTYPE html>  <html>  <head>  @RenderPage("../Shared/Header.cshtml")  </head>  <body class="gray" onselectstart="return false">  @RenderPage("../Shared/BodyLayout.cshtml")  <script type="text/lizard-config">  {  "url\_schema": "detail/{pid}/{scid}/{dcid}/{from}",  "model": {  "apis":[{  url:'http://m.ctrip.com/restapi/vacationapi/product/productdetail',  postdata:{"pId":Lizard.P("pid"),"scId":Lizard.P("scid"),"dcId":Lizard.P("dcid"),"reqType":[1],"head":{"cid":"a64ac7fd-203a-83bd-5e99-6bb4d594171b","ctok":"351858059049938","cver":"1.0","lang":"01","sid":"8888","syscode":"09","auth":""}}  }],  "filter":function (datas){  var ret = {data:datas[0].data, productType: ''};  return ret;  },  setTDK: function(datas)  {  var detailData = datas[0].data;  return {title: detailData.pName, description: detailData.title, keywords: detailData.pName}  }  },  "view":{  "header": Lizard.T("headTmpl"),  "viewport": Lizard.T("detail\_tab")  },  "controller": "@ConfigurationManager.AppSettings["WebresourcePDBaseUrl"]tour/controllers/detail.js"  }  </script>  <script id="headTmpl" type="text/lizard-template">  <h1>产品详情</h1>  </script>  <script id="detail\_tab" type="text/lizard-template">  @RenderPage("templates/detail.cshtml");  </script>  <script id="detail\_route\_tab" type="text/lizard-template">  @RenderPage("templates/detail.RouteTab.cshtml")  </script>  <script id="detail\_score\_tab" type="text/lizard-template">  @RenderPage("templates/detail.ScoreTab.cshtml")  </script>  @RenderPage("../Shared/Footer.cshtml")  </body>  </html> |

我们可以看到这个标签（<script type="text/lizard-config">）占了页面中的较大篇幅。因此有必要说说这个配置。这个配置定义了一个JSON结构。可以说，整个Lizard页面解析器的运转都是按照这个SCRIPT标签中的定义进行的，下面解释几个字段：

* url\_schema 制定URL格式 例如如下URL

<http://localhost:31818/vacationList/2/2/%E4%B8%8A%E6%B5%B7/%E5%91%A8%E8%BE%B9/1/nopage> 可以定义URL Schema为

"/vacationlist/{salecity}/{scity}/{sname}/{kwd}/{type}/{nopage}"

对于QueryString类型的参数无需以大括号的方式指定参数，框架天然支持。

* models 指定渲染此页面所需要的数据 这个对象包含如下属性
* api 后台服务地址，这个是一个数组结构。支持通过两个接口返回的数据组合后渲染模板。

postdata 后台数据post服务的数据，这里用到了框架API Lizard.P用来获取URL中传递过来的参数，这个API接受一个参数，对于路径中的参数，按照 urlschema中定义的参数名称获取。对于querystring中的参数，直接传入参数名称即可获取。

filter：对于后台服务传递过来的数据如果无法直接使用，可以在filter中定义处理后台数据的函数，filter输出的数据，会被框架用来做模板渲染

setTDK: SEO中用来渲染title description keyword

* view: 指定渲染模板，这里一般调用框架 API Lizard.T 这个API 接受一个参数就是模板区域对应的Id 另外业务原先写的模板HTML文件需要重命名为cshtml，通过@RenderPage引入到主页面中
* controller 对应之前每个View的Js，和以前一样也是一个AMD快，同时View模块继承的是cBasePageView替换为cPageView

## 几个问题的澄清(待补充)

* API的restful地址是否可以配置

参见http://conf.ctripcorp.com/pages/viewpage.action?pageId=50988089

* view的onLoad已经废除，请使用onShow。另外由于首屏渲染已经交给框架完成，所以view的turing和injectHeaderView已经废弃。
* 一些页面本身不需要请求数据比如首页，如何改造？

"model": {

"apis": []

}。

Models按照如上配置即可。一般这样的页面的模板是静态的，所以无需数据渲染。

* 数据来源不是restful接口

如果这个页面是SEO页面，且渲染主界面所需要的数据来源于自定义的模块。那么只能把这个接口修改为restful接口。如果是SEO页面的局部区域渲染需要这些数据，只能在定义URL schema的时候把这个信息在URL带入进来

如果这个页面不是SEO界面，参见下面的问题

* 对于不需要做SEO的界面如何改造

可以定义config中的model如下

"model": {

"apis": []

},

同时对于ViewPort的模板区域不指定任何模板

<script id="viewportTmpl" type="text/lizard-template">

</script>

这个时候框架什么都不会做，各业务在onshow中按照之前的方式通过model取数据后渲染

* 页面中存在局部信息的渲染依赖用户操作行为，如何解决?

这些信息通常是SEO 不需要的，因此可以在onshow的代码中通过model获取数据后在进行渲染。

* Liazrd.showDiaLog

这个 API被创建出来用来实现页面被一个特殊的“模态对话框”覆盖，但是不会产生History记录。

这通常对应两个业务场景

1. 仅仅是视觉上的一个新页面，但是数据是之前已经准备好的
2. 需要重新请求数据（最典型的是 SEO下的分页数据），但是无需产生History记录

这个 API接受两个参数第一个参数和lizard-config中的 JSON格式定义一致，第二个参数可以不传，是一个对象，指定Dialog名称以及是否pushstate，第二个参数 不传，则框架会生成对话框名称，并且按照不pushstate处理。

样例代码：

|  |
| --- |
| var viewParams = {  "model": {  "data": params,  "filter": $.proxy(filterCallback, self)  },  "view": {  "header": "", //optional  "viewport": Lizard.T(template)  },  "controller": Lizard.appBaseUrl + 'webresource/tour/controllers/vacationlist.js'  }  }  Lizard.showDialog(viewParams); |

* 由于路径不再是主页+Hash的格式，每个页面的路径会不一样，因此requireconfig请使用绝对路径，参见如下代码

|  |
| --- |
| require.config({  paths: {  'TourModel': Lizard.appBaseUrl + 'webresource/tour/models/tourModel',  'TourStore': Lizard.appBaseUrl + 'webresource/tour/models/tourStore', |

* Runat属性

对于某些元素，只有SEO页面才有，比如 懒加载表格，在SEO页面为让爬虫正常获得翻页数据，需要在SEO页面渲染“上一页”，“下一页”按钮。而这些按钮在用户浏览页面是无需渲染的。



参见上图，由于是SEO页面，所以相关的“按钮”也是以A标签的方式渲染

* lizard-catch="off"

之前的页面，页面间的跳转，通过事件触发。但是这样的设计，不便于搜索引擎爬虫抓取。所以需要在原界面增加A标签，点击A标签，事件会冒泡到原绑定事件的DOM元素上，由于原页面已经绑定了事件，为防止这种”重复”行为的发生，可以在为 SEO增加的A标签上增加lizard-catch="off"，这样用户点击的时候，会防止A标签的默认行为发生。

* 模板嵌套能力，在模板中允许调用Lizard.T API来实现模板嵌套

示例代码如下：

|  |
| --- |
| <div class="home\_base\_loading base\_loading"><i></i>加载中<%=Lizard.T('inside\_tpl', {data: {a:'this is an inner tpl'}})%></div>  …………  <script id="inside\_tpl" type="text/lizard-template">  <%=data.a%>  </script> |

* 获取localStorage能力API

第一个参数是localStroage的名称，第二个字典是LocalStoage对象的字段，最后一个是默认值

Lizard.S(stroename, key, defaultvalue)

# 主要迁移步骤

* 规划自己频道的 URLSchema
* 安装.net MVC3
* 新建.net MVC3项目
* 修改global.asax文件，按照URLSchema定义路由规则。
* 在Controler中根据global中定义的路由规则，创建Crtoller文件（C#）代码。
* 在View中定义每个页面的cshtml,主要是lizard-config配置
* 将公共的Footer和header拷贝到自己的项目中（除页面Controller其余的JS和CSS资源不要写在各页面中，统一写在Footer和Header中）



其中PD自定义的CSS可以追加到header中，业务自定义的requireConfig在Footer中设置

* 原先的View的JS代码拷贝到指定目录（根据动态页面中lizard-config制定的controller目录）。
* View JS引用模块从cBasePageView替换为cPageView，删除对渲染页面的静态HTML模板的加载依赖。删除this.turning this.injectheaderview调用
* 部分页面局部信息的渲染依赖用户操作行为，用户位置等，按照第5节描述的方式解决。

# 本地V8引擎渲染调试

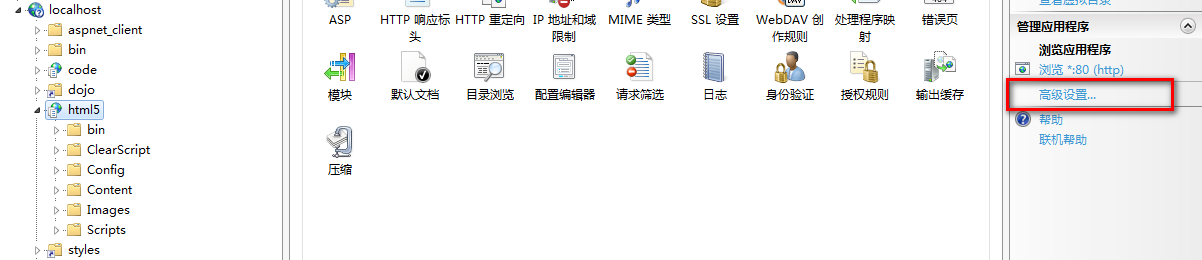
基本原理：IIS需要部署.net V8程序到/html5目录，部署web程序到/webapp目录。当访问 html5站点时，.net V8渲染程序会替换UTL中的HTML到 webapp获取对应的页面以及JS资源，并调用V8引擎完成页面渲染后，作为response返回给浏览器。

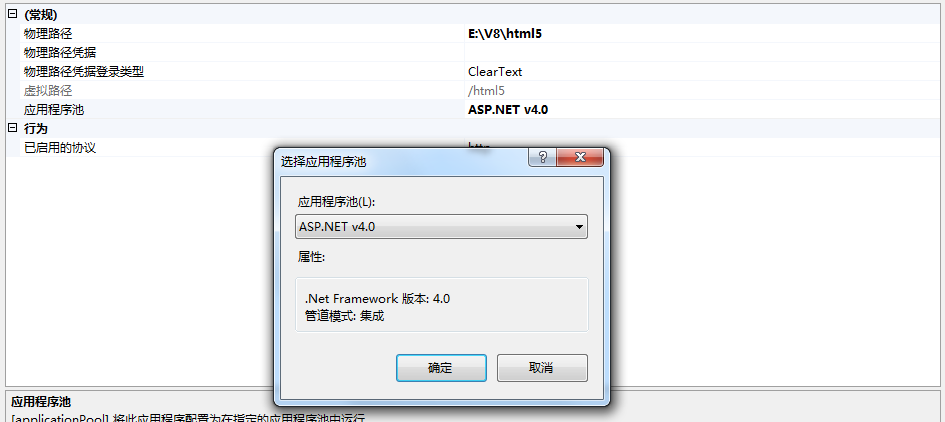
环境准备：

首先需要安装微软的Visual C++运行时库



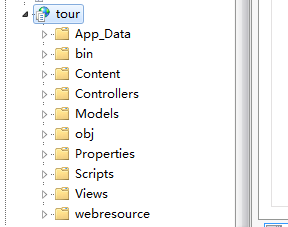
将Html5.zip解压后部署到IIS上，并将应用程序池设置为ASP.net v4.0如下图





将Lizard 2.0 web项目部署到IIS 的webapp目录下，并设置应用程序池设置为ASP.net v4.0

如下图



部署完毕即可访问，以团队游为例

本地调试SEO页面链接

<http://localhost/html5/tour/vacationlist/2/2/%E4%B8%8A%E6%B5%B7/mobi/1/nopage>

本地调试WEBAPP页面链接

<http://localhost/webapp/tour/vacationlist/2/2/%E4%B8%8A%E6%B5%B7/mobi/1/nopage>

# Web方式的打包

由于 Lizard2.0方式是多页面方式，所以web下，仅对项目JS做压缩和混淆。

# App 以及打包

## App打包过程

App打包样例配置如下，其中：

hostname表示 web工程部署的IIS服务器地址

views表示主要页面URL地址（页面参数可以随意填，只要保证页面访问不会抛出.net或者HTTP错误）

defaultview表示首页

channel: 表示频道名称

|  |
| --- |
| {  "hostname":"172.16.140.161:8082",  "views":{  "/webapp/tour/index":1,  "/webapp/tour/VacationList/2/2/%E4%B8%8A%E6%B5%B7/mobi/1/nopage":1,  "/webapp/tour/Detail/1666873/2/2/nopage":1  },  "defaultView":"/webapp/tour/VacationList/2/2/上海/mobi/1/nopage",  "channel":"tour"  } |

App下，Lizard2.0会通过打包的方式，将多页转换为单页，也就是只有一个index.html首页，其余的资源都做成AMD模块被引用。打包的过程如下

* 首先将待打包的频道工程（.net）项目发布到IIS上（可以是本地IIS）；
* 在打包配置中配置每个页面的地址（页面参数可以随意填，只要保证页面访问不会抛出.net或者HTTP错误）。
* 打包程序会分析拉下的静态View HTML资源，并修改原文件对于web资源的应用路径（包括CSS JS等）调整线上地址为本地地址，requireconfig地址也会做相应调整。
* 拉下资源会按照打包配置中的 views配置项，生成对应的目录以及静态资源。
* 生成本地路由文件LizardLocalroute.js
* 生成的文件，目录结构为 频道名称/web IIS服务器名称（无端口）/资源对于web根路径地址。

## App本地路由

打包的时候，打包程序会读取每个页面的urlschema配置作为本地路由表的key,同时将view的路径作为路由表的 value。参加团队游打包样例文件，可以看到这些 value就是本地html路径地址。

当调用Lizard.goTo的时候，如果是Hybrid环境，框架迭代URL中的urlschema 生成正则表达式去和送过来的url匹配。如果匹配到，则将对应的 html的路径返回作为requirejs 的输入。获得文本后解析lizard-config配置完成对view的渲染并执行controller。

## 打包操作步骤

将开发后的频道工程发布到IIS服务器

按照工程的具体情况配置打包配置文件，样例如下



打包程序依赖jsdom，而jsdom依赖python,所以打包机器如果没有安装python, 请先安装python。

下载框架提供的打包 node js程序



将这两个文件 copy到一个目录下 ，执行如下批处理文件

|  |
| --- |
| if not exist .\node\_modules (goto install) else (goto package)  :install  call npm install colors  call npm install vm  call npm install url  call npm install path  call npm install mkdirp  call npm install jsdom  call npm install CSSOM  call npm install util  call npm install esprima  call npm install os  echo "npm intall completed"  :package  node busbu.js --path=busbu.json |

从以上红色代码可以看出，配置文件的和node运行的 js代码的文件名称以及路径可以自己指定。