**前端基础架构设计文档概要**

|  |  |  |
| --- | --- | --- |
| **文件状态**  [√ ] 草稿  [ ] 正式发布  [ ] 正在修改 | 文件类别 | 开发文档 |
| 文件编号 |  |
| 当前版本 | v 1.0 |
| 完成日期 |  |

**内部文件**

说 明

本文档为茂松科技有限公司文件，资料信息归茂松科技有限公司所有。

**修订记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **修订版本** | **作者** | **参与者** | **日期** | **修订说明** |
| v 1.0 | Lyle(陈维汉) |  | 2014/07/09 | 初稿 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**目录**

[1：文档介绍 4](#_Toc3815)

[1.1 文档目的 4](#_Toc19951)

[1.2 文档范围 4](#_Toc28417)

[1.3 阅读对象 5](#_Toc29884)

[2：名称界定 5](#_Toc24167)

[3：设计流程 5](#_Toc20421)

[3.1 目录结构 5](#_Toc21438)

[3.2 流程图 8](#_Toc22025)

[4：功能模块 9](#_Toc8498)

[4.1 静态CDN目录 9](#_Toc5003)

[4.1.1 js基础库与组件库 9](#_Toc9014)

[4.1.2 默认css与个性化站点 9](#_Toc22911)

[4.1.3 默认img与个性化站点 9](#_Toc31351)

[4.2 模块加载器 9](#_Toc12546)

[4.3 模块配置 10](#_Toc21049)

[4.4 模块聚合 10](#_Toc11388)

[4.5 page模块 10](#_Toc32430)

[4.6 打包压缩 10](#_Toc21783)

[4.5 页面版本发布 10](#_Toc23313)

[5：代码规范 10](#_Toc7413)

[6：实际问题解决 11](#_Toc19846)

[8：PC的优化和整合 11](#_Toc17415)

[9：架构迭代 11](#_Toc18194)

[10：配合人员与建议 12](#_Toc19641)

# 1：文档介绍

## 1.1 文档目的

根据目前项目实施的需要和已有项目的优化和调整，为了系统运行的稳定，代码质量的控制，后期便利维护，以及后续参与者尽快的参与进来的目的，特编写该文档。（以下为一个初步文稿）

## 1.2 文档范围

本文档仅是开发文档，涉及系统的实施开发相关的程序，流程,框架，模型，组件。

## 1.3 阅读对象

本文档适合参与项目的开发人员阅读，也适合从事跟项目相关工作的人员阅读。

# 2：名称界定

前端基础架构整体 英文简称：swift。雨燕是飞翔速度最快的鸟类，常在空中捕食昆虫，翼长而腿、脚弱小。我们前端追求 又快又轻，取名swift.

# 3：设计流程

## 3.1 目录结构

描述：根据目前公司现状，主要是把前端剥离成两大块。

第一块是分解资源和非业务逻辑绑定的静态文件，进行CDN缓存，使用全路径域名引入，资源放置可以随意迁移，而且非业务逻辑的不存在跨域的问题。

第二块绑定业务逻辑相关的，（包括数据请求的，经常变动的数据操作）均独立出来跟随站点放置。

讨论HTML视图层需要抽出独立共用吗？

我个人建议：不需要，界面渲染给逻辑相关。如果抽出共用，后期维护个性化的东西多了，太多判断(判断需要JS介入)，成本成倍增加，毕竟HTML不像JS可以进行编程。聚合比较高了，灵活性相对降低。

**swift目录**

**|--mobile**

**|--css**

**|-- defalut (站点通用样式)**

**|-- weisiteA (站点个性化样式)**

**|-- ........**

**|--js**

**|-- base (站点基础库，依赖库)**

**|--LAB.js（资源加载器)**

**|--zepto.js（DOM操作基类）**

**|--until.js(自定义工具类)**

**|--JSON2.js（依赖工具类库）**

**|--ejs.js (js模版引擎)**

**.......**

**|-- componnet（站点组件库）**

**|--imglazyload （图片懒加载组件目录）**

**|--zepto.imglazyload.js （组件）**

**|--zepto.imglazyload.html（组件使用方法demo）**

**|--img (组件图片)**

**|--css（组件CSS）**

**|--slider（幻灯组件目录）**

**|--.....**

**|--img**

**|-- defalut (站点通用图片)**

**|-- weisiteA (站点个性化图片)**

**|-- PC (结构类似)**

**站点目录**

**weisiteA 目录**

**|--html**

**|--index.html**

**|--news.html**

**|--js**

**|--config.js(资源配置文件，资源引入全局变量，版本号管理，压缩打包配置等)**

**|--index.js(index.html映射以的页面为单位的业务逻辑类)**

**|--news.js（news.html映射以页面为单位的业务逻辑类）**

同时也会映射引入相对应的index.css与news.css。

前后端交互以AJAX为主，更新版本号需要重新生成CDN缓存。

样式改变有2种使用方式：

一：删除引入，重新引入新样式实现

二：引入默认样式，在引入新样式重载样式实现

大同小异的站点，我建议采用第二种方式? 减少浏览器回流

Css共用样式的写法更多参考bootstrap。

Eg: <img src="..." alt="..." class="img-rounded">

<img src="..." alt="..." class="img-circle">

<img src="..." alt="..." class="img-thumbnail">

## 3.2 流程图

简单流程图：

资源统一规范管理，引用，组件的积累重用。

越来越大CDN资源库swift ，非CDN资源库。

Config配置文件,获取需要的资源

根据需求按需灵活配置资源。

统一动态加载管理，省去人工引入，方便管理。

加载器针对不同的页面进行路由分解，获取配置文件的资源进行加载，并调用页面模块初始化事件。

# 4：功能模块

## 4.1 静态CDN目录

### 4.1.1 js基础库与组件库

基础库提供基础的基类库，加载器，md5等

组件库提供通用或者个性化的组件。

### 4.1.2 默认css与个性化站点

默认的CSS

个性化站点需要先引入默认CSS继承，然后重载样式，修改。

完全独立的站点可以不引入默认的CSS，引入自己的CSS，响应。

### 4.1.3 默认img与个性化站点

默认的Img（需要压缩优化）

个性化站点图片

平板与手机差距比较大，是否需要根据不同的屏幕大小调用相对应大小的图片设计？实际过程在讨论也行

## 4.2 模块加载器

兼容PC与移动设备的JS与CSS 加载，有先后加载的顺序管理。

加载过的页面都会进行缓存，下次请求不会在发送http请求。

## 4.3 模块配置

提供全局变量【模块变量，浏览器信息（大小，分辨率，系统等），ajax是否开启缓存，资源路径全局变量，API请求路径前缀，版本号，类型（开发环境，生产环境）等】

提供按需加载的配置JSON对象

提供手工路由分解URL【自动路由需要一一对应的映射关系】

提供调用加载器加载数据及触发模块初始化。

## 4.4 模块业务逻辑聚合

如果在业务逻辑代码开发过程中，存在公用代码只能局限与该站点，不能提升到通用库里，可以在站点内模块聚合。

## 4.5 page模块

Page模块，结构模版类，提升代码质量和管理，维护更方便。

## 4.6 打包压缩

针对是开发环境还是生产环境的类型，调用非压缩与压缩代码。

打包压缩目前采用手工分部分，合并压缩.

压缩的时候可以按照页面需要进行压缩，尽量减少不需要的代码合并压缩。

CSS的图片引用存在路径问题，不能盲目合并打包。

有些组件有图片和样式的，也存在这个问题。

## 4.5 页面版本发布

提供了新版本功能上线，用户不需要手动清楚缓存的功能，前端就调用最新代码

# 5：代码规范

1：命名的语义化

2：参考代码规范文档

# 6：实际问题解决

1：多个站点，样式上大同小异，是怎么处理的？

以默认模版为基础，CSS进行重载修改，JS进行策略

2：目前代码调用，代码书写质量都比较差，是怎么处理的？

结构规范化，模块规范化，组件规范化，约束了参与人员的随意性。

3：目前资源，组件调用也很混乱，是怎么处理的？

资源统一管理，使用加载器统一加载，组件也一样。

4：代码优化做了那些？

统一的CDN缓存，组件大量复用，后期压缩合并，减少http请求等

5：多设备，多分辨率？

识别设备，分辨率，JS进行策略，CSS使用media

7：浏览器加载资源的缓存

浏览器的资源请求，使用了缓存基本上是两种情况

status code: 200 ok ( from cache )

status code: 304 Not Modified

第一种方式是不向浏览器发送请求，直接使用本地缓存文件。

第二种方式，浏览器虽然发现了本地有该资源的缓存，但是不确定是否是最新的，于是想服务器询问，若服务器认为浏览器的缓存版本还可用，那么便会返回304。

使用的第一种，减少请求，更新由版本号来控制。

6：增加代码版本号，代码更新，浏览器及时显示的提升用户体验。

# 8：PC的优化和整合

　　　 在mobile站点开发周期结束后，就开展PC的优化工作，部分页面需要重构，部分JS,CSS都需要整理，以融入 swift里。先优化模版站点，然后进行下一个站点的优化，最后以达到统一管理所有站点的前端。

# 9：架构迭代

目前的架构是由 加载器（模块就是page） + jquery/zepto（DOM操作） + page模块（即页面控制器）+ HTML/ejs(视图) + JS组件（以Jquery/zepto 插件方式） 的组合方式。

Model层这里指后台提供过来的标准的数据。

后期稳定后，架构会向Seajs/Requirejs（模块加载器） + Backbone.js （MVC）+ jquery/zepto （DOM操作）+ HTML/ 模版（视图）+ JS模块（即JS组件AMD或者CMD的规范方式写的） 的组合方向去迭代。

# 10：配合人员与建议

人员：

linux系统运营维护人员

PHP API 后台人员

IOS，android开发人员

设计师

建议：希望能招聘或者借调2-3个前端初中级开发工程师过来，组建前端团队，后期工作量是很大的。

架构的问题修正及维护调整，组件开发，日常代码的质量管理，后台数据的交互，HTML+CSS页面的构建，多设备兼容性，多分辨率兼容的调试开发（苹果，android,平板 涉及为页面布局与组件功能）。

注：1：运维需要提供CDN需要支持单个文件或者目录更新的实现。

2：内部网络的问题，手机连接wifi可以访问有线电脑本地服务器ip地址。（优先）