# Buildjs

# ——基于Seajs的Web前端模块化架构

# &编译器

**【设计思路】**

By pakinguo

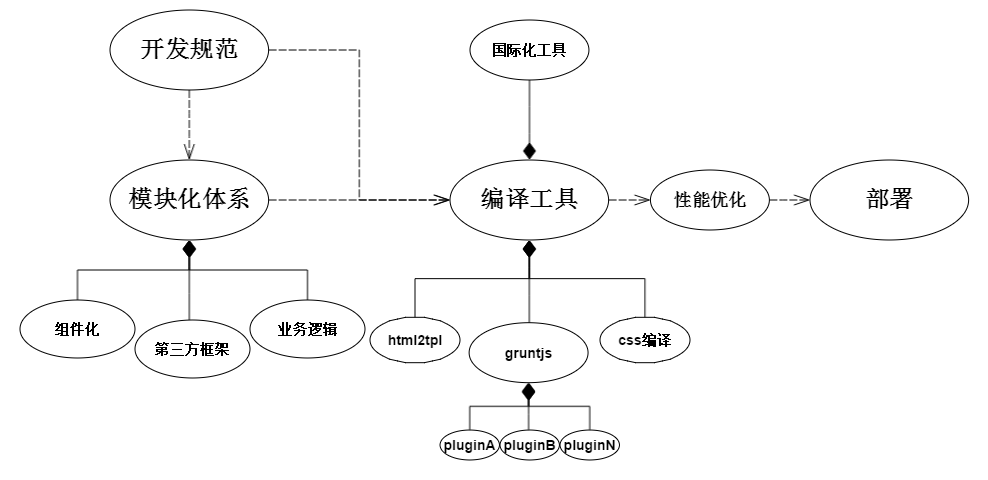
## 前提

需要了解seajs与gruntjs，参考官方网站：

Seajs：<http://seajs.org>/

Gruntjs：<http://gruntjs.com/>

## BuildJS体系示意图



## BuildJS设计思路

#### 为什么需要BuildJS

随着项目业务逻辑的增加，先前的开发方式已经多少会对效率产生一定的影响。同时，对于旧页面和模块的复用，大部分方式是ctrl+c & v，而不是真正意义上的引入组件配置化。以及，对于前端性能的优化，旧项目基本没有涉足，仅仅做到的小部分，只有在移动web页面针对菲律宾马来等网络速度不是很快的国家，才有针对性地手动合并压缩静态资源文件。由于大部分项目是一人责任制，比较少有合作开发，故命名空间的污染问题基本没遇到。但是随着项目进展，迭代的快速开发，合作开发已经开始增多。

现状是，web前端在js上除了用jquery、zepto以及knockoutjs、backbone、underscore等比较多的针对DOM操作或渲染模板的第三方框架外，没用其他管理模块或管理整个架构的第三方框架或编译器。这样，虽然对于快速迭代开发来说效率依然保持较高的状态，但是，没太多的管理工具会导致在维护或者代码合并压缩上有很大的障碍。通用组件的复用上会因为修改了某一个文件，导致另一个复用相同代码的组件也需要同时修改，而不是在其基础上进行扩展修改。

配合php端使用的smarty，前端的tpl、js、css等开发时是分别放置于不同的目录下：tpl放置于views目录，而js、css等放置在resource目录下，同时，views与resource同级。当业务对应的页面不是很多的时候，没多大问题会暴露出来。但随着业务的增多，resource和views两者间在IDE或者文本编辑器上就很难在一个屏幕内轻松的找到对应的模块了。而且，当一个bug出现，需要快速定位修复时，页面和js、css来回查找，一定程度上影响了效率。旧的目录规范已经不太适合日益增长的业务模块了。

没有自动化工具可以来合并压缩指定的文件，或者进行国际化翻译代码，基本都是手动处理，浪费时间浪费精力，前端性能优化有待提高。

基于以上的问题，需要采用新的开发模式来提高开发效率、前端性能。首先从requirejs和seajs中选择了seajs，原因是比seajs更易上手，且跟nodejs保持一致的cmd规范，对seajs开发的项目进行构建编译有比较强力的工具后盾。接着，在文件目录规范上需要做调整，以模块为单位，将所有静态资源文件和页面都放在同一个模块命名的文件夹下，再用子文件夹区分。然后，需要对构建工具gruntjs进一步进行封装，提供自动化与手动化两种方案用于构建项目。最后，需要做国际化支持，这跟新的目录结构规范也相呼应。

那么问题来了，BuildJS到底需要解决什么问题？

#### 解决问题

1. 制定新规范，统一Seajs模块框架的目录规范和开发规范，达到开发、测试、产品环境的代码可快速切换，并可做静态资源版本化控制；
2. 帮助前端开发分离业务、页面和组件，降低js、html与css的耦合度，；
3. 开发业务或组件时，专注在模块中编写代码（包括js、css、tpl以及自定义的html片段，如.htpl），有工具来做实时同步和文件转化；
4. 开发后，代码部署到测试环境时，需要对代码进行合并压缩，提供自动化编译方式和手动配置化编译方式；
5. 支持可提取代码中需要国际化的字符串，并根据翻译好的文档（先支持po文件）按语言类型进行翻译。

#### 目录规范

|  |
| --- |
| project  ├ front  ├ conf（开发时关注的配置文件目录）  ├ lib（基本上是增量添加第三方库，可考虑按需放到CDN）  ├ \_\_src（开发时关注的模块目录）  ├ app  ├ common  ├ normalize.css  ├ common.css  └ common.js  ├ login  ├ css  └ index.css  ├ img  ├ icon.png  ├ index.png  └ ...  ├ js  └ index.js  └ view  └ login.tpl(运行工具后，不会同步到src下，而是同步到views)  └ widget  ├ ivy.alert  ├ ivy.alert.css  ├ ivy.alert.js  └ ivy.alert.htpl (--> ivy.alert.htpl.js 通过编译工具)  ├ src（实时同步从\_\_src生成而来的开发环境运行的文件，目录结构与\_\_src一致）  ├ en-US（由国际化工具生成的）  └ zh-TW（同上）  ├ views（有工具同步front下的\_\_src中的view层文件到此目录，只保留两层目录结构）  ├ src  ├ common  ├ head.tpl（只保留两层目录结构：模块名/文件名.tpl）  └ foot.tpl  ├ index  └ index.tpl  └ login  └ login.tpl  └ en-US  └ i18n  └ i18n.po |
|  |

1. Web前端关注两个文件夹：front和views；
2. conf项目的配置文件，配合实时同步工具，可以按项目生成不同的配置文件；
3. lib放置第三方框架，包括未cmd化与cmd化，框架间用版本号区分；

对于2、3点举例子，Seajs的配置seajs.conf.js：

seajs.config({

base: "/cms/resource/",

alias: {

"jquery": "lib/cmd/jquery/2.0.3/jquery.js",

"jqueryui": "lib/cmd/jquery/jqueryui/1.11.0/jquery-ui.min.js"

},

debug: false

});

lib下的文件目录

|  |
| --- |
| projects  ├ \_\_src  └ lib  └ cmd  └ jquery  ├ 2.0.3  └ jquery.js  └ 1.11.0  └ jquery.js |

1. app是页面逻辑业务模块，以页面为单位建立文件夹，其子文件夹结构可以灵活变动，但建议按照上面提供的目录结构方式，有助于统一规范；
2. widget是通用组件层，一般是其他页面需要用到的通用的模块抽离而出，可通过编译工具输出单文件形式（即是合并js、css、htpl，但由于css里的相对路径编译为内联后会有路径找不到的问题，有待解决）；
3. 前端开发html、css、js都分离开来，利用定制的自动化工具，为自定义的某些后缀名html模板文件（如.htpl）编译成seajs模块文件或其他适用的js文件，配合模板引擎工具如[backbone](http://www.backbonejs.org/)、[underscore](http://underscorejs.org/)、[mustache](https://github.com/janl/mustache.js)、[HandlebarsJS](http://handlebarsjs.com/)等来进行前端渲染；
4. 国际化，包括front下的src和views下的src（文件后缀默认为html、js、css、tpl等），以文件夹命名语言来进行国际化（有别于本地化，本地化部分逻辑上需要调整，国际化只是翻译问题，关于本地化有待进一步考虑方案）；
5. 环境切换，开发、测试环境如何快速切换？只需要修改seajs.conf的base属性即可，这样可以做代码的快速切换和测试。

#### 实时同步代码

根据上面的目录规范，实时同步代码的功能解决下面的问题：

\*备注：由于seajs加载css的局限性，故自主开发了seacss用于在页面同步加载css文件

1. 由于配置文件也是通过几个部分合成，将seajs配置文件进行分离（参考seajs.config方法），主要是将配置项“alias”独立成一个json文件来维护，同时针对seajs和seacss，分成\_\_jsalias.json和\_\_cssalias.json两个文件来配置alias属性，再根据\_\_seajs.conf.js的模板文件来最后生成的页面使用的配置文件，建议放置于conf目录下，一般不用多次修改模板文件，而是修改.json文件。
2. 将代码从\_\_src同步到front下的src和views下的src，这一步是分离静态文件和php渲染的tpl文件。这是由于\_\_src只在开发的时候可见，测试、发布的时候是不会有这个文件夹的存在，故需要把模块中的静态资源和页面资源分离开。views下只同步.tpl的文件，并保留最多两层目录结构（扁平化，主要是根据模块来区分页面，所有业务或功能，都是以模块为最基本单位，即使通过front下的\_\_src的app再封装几层目录结构，也是以view目录为基准，取其父文件夹模块名和子文件.tpl的文件名为views下的两层目录结构）。
3. 为了更好地分离js和html的内联，自定义了.htpl的文件类型用于编写html代码片段，主要是把js里可能涉及到编写较长的html的片段提取出来，通过工具编译为seajs模块，通过require方式可以方便加载（开发时会多一个请求），最后再通过构建工具合并成一个文件（测试、发布时就不需要多一次请求）。
4. 在Linux下，同步的文件读写权限和用户组并非一致，需要进行修改，只需要调用shell命令。

#### 构建（编译）

需要对开发版的seajs模板进行构建，以降低其在未transport前需要多进行的模块解析和缓存，以及对模块进行合并压缩，减少http请求。

对seajs模块进行transport，需要配合gruntjs来完成。Buildjs构建工具的思路是根据输入的待构建的路径，先生成gruntjs需要的构建配置文件，再调用shell执行gruntjs命令进行构建。另外，由于第三方框架不参与构建（第三方框架由于数量多且代码文件较大，会影响构建花费的时间，测试中发现构建10多个第三方框架需要的时间是构建项目业务模块的10多倍时间，而构建全部业务模块一般需要5~10秒），编译工具会先排除指定的lib或默认的lib文件夹，再进行构建。

Transport后的模块需要下一步动作：uglify——合并压缩js或css文件。目前暂时只完成合并压缩js文件。其实现思路跟transport大致一样，只是多了一个步骤——需要根据模块去查找其相关的模块依赖链，再根据这些依赖链提取需要合并的文件列表，生成gruntjs需要的配置文件，执行shell。默认自动合并压缩的功能是以单一模块为单位，即是以文件为单位，模块中有多少个依赖文件，就以这些依赖文件作为合并压缩的文件列表。可传入参数指定需要忽略合并的模块，比如第三方框架不需要合并，或者通用组件不作为合并的文件（由于通用组件可以在不同页面使用，只用某个页面使用过，访问其他页面会从缓存中读取该通用组件的代码，减少重新请求），这样就达到优化部分前端性能的要求了。

另外，如果有需要，可以直接配置gruntjs的构建配置文件，进行手动构建。目前构建工具只做全量构建，不做单一文件的构建。

#### 国际化

根据po文件的格式，利用nodejs编写一套提取工具，按照约定的标记先在代码中标记号需要翻译的字符串。由于国际化中对需要翻译的字符串进行的标记提倡的是不影响代码的正常运行，故需要对标记做完善的正则适配提取。提取之后按照po文件格式生成未翻译的po文件，交给翻译组翻译后，再应用国际化工具，对做了标记的代码进行翻译，比如从src文件夹生成到en-US等。之后对再对修改过的代码重新提取，翻译工具会进行去重，已翻译已提取的不再提取，只记录重复出现的文件对应的行数。

## Buildjs思考中尚未实现的功能

1. 对css按某种格式进行合并压缩，包括可以加注释// @require xxx.css或是直接用@import xxx.css方式来合并压缩；
2. Css转为seajs的内联调用方式后，会有图片相对路径引用的问题，如何无缝解决；
3. 包括可以用base64将小图片压入css中；
4. 其他未知