一种跨技术栈的前端国际化方案

电子超市开发部（友云采） 宋江凌

灵感来源

大多数项目在开发伊始都采用小步快跑的方式进行快速迭代上线，甚至于不同的工程采用了相互不兼容的技术栈进行开发。如果要让这些不同技术栈的项目都支持国际化，会是一种怎样的体验？

- Requirejs + knockoutjs + jquery
- Webpack1.x + knockoutjs
- Webpack2.x + vue
- Webpack1.x + backbone + vue
- 还有一些是完全没有工程话化的静态页面

技术栈完全不同，所以现有的一些开源的国际化方案：

- jquery-i18n
- Knockout-i18n
- React-i18n
- Vue-i18n
- 其他等

我们可能需要抛弃掉（因为在项目众多资源有限的情况下，不可能一个个去验证）

运行时翻译和编译时翻译

比较主流的一些国际化方案都是采用运行时翻译（webpack-18n 则是运行时编译的方式）。二者最大的不同在于：运行时编译强依赖与你当前使用的开发框架，如 react 你需要使用与 react 配套的 i18n 工具，vue 需要使用 vue-i18n 或其他配套工具；而编译时则原则上是使用特定的工具，在编译过程中对需要翻译的代码进行替换。

总结来说运行时翻译就是：

1.抽取词条

2.形成资源文件

3.对代码中原有词条使用特定的占位符或标识符进行替换

4.运行时根据当前上下文进行读取对应词条进行替换

而我最终不采用运行翻译的原因主要在于步骤 3.

1.在步骤 3 中如果通过人工纯手工来替换，那么人工成本无非是巨大的，还容易引发代码出错

2.如果是使用工具来抽取，那么很显然针对不同的技术栈需要研发与之匹配的工具，那么成本也是不小的

3.同时由于无法决定用户是用了哪些第三方插件，可能会导致第三方的插件的国际化适配也会增加困难

4.抽取后由于没有了原来的中文词条，维护人员对代码理解起来也会产生困难

核心：替换

国际化的核心在与：

1.多语文本的替换

2.组件对于内容文本变化后的自适应

其中第二点我们交给组件去做了（比如友云采 FED 自研的基于 ko 的 ui 框架 ycloud：https://github.com/yonyouyc/ycloud）。所以我们自动化的核心就留在了**多语文本的替换**上

现有的前端资源文件无非是.js，.html，.css，.vue, .ts 等等，而这些文件的内容无非就是一行行的文本，所以我们的思路就转变：

1.抽取工具

2.抽取中文生成 excel

3.通过一些翻译接口进行自动翻译或者交给专业的翻译部门进行翻译

4.使用替换工具把翻译好的 excel 对文件内容进行替换

抽取工具

抽取的关键有两个

1.抽取中文

在这种场景里其实正则表达式并不一定是最好的选择；抽取是针对每一行进行抽取，而每一行中可能同时存在多个中文：

如：html 中 （姓名：{{name}}（全职））

很显然上看最终抽取出的词条应该是有两条：姓名、（全职），所以我们最终选择了按字符读取的方式，毕竟抽取过程我们不需要太考虑性能问题，而实际实践发现抽取一个 5000 文件的工程基本 200ms 就可以完成.

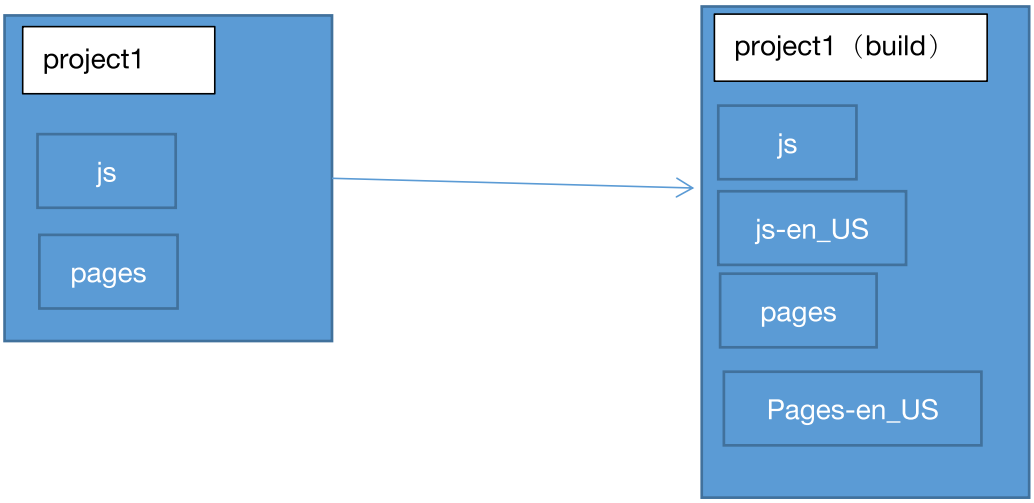
2.忽略注释

单行注释处理起来很简单，前端的注释无非 html 的注释和 js 的注释，而多行注释则需要在抽取过程中设置一个变量来存储多行的开始和结束。

考虑到了以上两点，把所有词条存储成一个 json 最终再调用第三方库导出成 excel 就可以了。

思维转换

承接替换第 4 步，这里要注意，我们原来是一套代码，而现在可能是针对一套代码产出了多套代码：



如上图所述，开发的时候还是 js 和 pages 代码，而编译之后会同步产出 js-en_US 和 pages-en_US 的英文代码，只需要在页面路由配置里加一些小小的修改就可以啦（毕竟识别当前上下文的代码不管哪种方案都是需要写的）。

以 requirejs 为例，我们只需要在 text.js 插件和路由定制的地方加两行小小的代码就可以了。

require 的 text 插件

```
load: function (name, req, onLoad, config) {
  var lang = Cookie.get('u_locale')
  if (lang && lang.indexOf('en_US') >= 0) {
    name = name.replace('pages/', 'pages-en_US/')
    name = name.replace('js/widgets/', 'js-en_US/widgets/')
  }
  // 其他语种也类似处理即可
  // origincode
」
```

同样路由切换的时候：

```
var lang = Cookie.get('u_locale')
if (lang && lang.indexOf('en_US') >=0) {
 module =  module.replace('pages/', 'pages-en_US/')
}
requirejs.undef(module);
$('#popup-content').html('');
require([module], function(module) {
```

扩展：单页面、多页面有无 webpack 编译，实质上都是一个思路去做即可，多页面可以统一嵌入一段公共的 js 即可。

替换工具

复用抽取工具的代码，按行进行文本 replace 即可。坑和需要注意的点：

1.为提高效率，在动态替换的过程中，需要读取每一个文件并把当前文件中的中文提取出来作为 key，方便减少替换和搜索范围

2.替换过程中需要先对中文 key 按长度做一个排序，原则上先替换长文本再替换短文本（原因：一个文件中同时存在：我知道，我知道你为什么这么帅两个词条），如果文件内容先把“我知道”替换成了“I know”，那么“我知道你为什么这么帅”这个词条就变成了“ I know 你为什么这么帅”，很显然可能就无法替换了

3.中文 key 可能存在“xxx 的手机”，而翻译的时候可能会翻译成 “xxx‘s phone”。因为有单引号的存在，如果恰好在 js 中使用了单引号，可能代码就报错了哦，同理还有双引号。这些都是需要大家注意的。

4.每次替换以前都需要自动重新执行一次抽取，并将抽取结果和翻译结果进行比对，以防止后续开发产生新的未翻译的词条。

实践

目前用友云采团队最核心的几个工程已经使用这个 i18ntranslator 工具完成了国际化前端的替换，效果显著，减少了大量工作量，对原有代码不存在 breaking change，且不需要团队开发时依照某种规范取严格执行的。

目前我们的工具已经开源并放到了：https://github.com/yonyouyc/i18ntranslator 欢迎大家了解试用并提意见，也欢迎邮件和我沟通交流（songhlc@yonyou.com）