**SANY国际化编码规范**

用友网络股份有限公司

二〇一八年八月

目 录

[1 国际化架构简介 3](#_Toc522113513)

[1.1 技术栈说明 3](#_Toc522113514)

[2 国际化工具使用说明 4](#_Toc522113515)

[2.1 国际化工具处理逻辑 4](#_Toc522113516)

[2.2 国际化工具的配置 5](#_Toc522113517)

[2.3 国际化工具的使用 6](#_Toc522113518)

[3 国际化规范 8](#_Toc522113519)

[3.1 规范原则 8](#_Toc522113520)

[3.2 统一规范要求 9](#_Toc522113521)

[3.2.1 资源目录结构规范 9](#_Toc522113522)

[3.2.2 资源文件命名规范 10](#_Toc522113523)

[3.2.3 资源内容定义规范 10](#_Toc522113524)

[3.2.4 其他国际化规范要求 11](#_Toc522113525)

[3.3 国际化管理规范要求 11](#_Toc522113526)

[3.3.1 语种的设置要求 11](#_Toc522113527)

[3.3.2 多语资源预制要求 12](#_Toc522113528)

[3.3.3 多语资源扩展要求 12](#_Toc522113529)

[3.4 数据多语处理规范要求 12](#_Toc522113530)

[3.5 基于React的国际化规范要求 13](#_Toc522113531)

[3.5.1 React-Intl国际化步骤 13](#_Toc522113532)

[3.5.2 国际化资源文件管理 14](#_Toc522113533)

[3.6 基于后台服务的国际化规范要求 15](#_Toc522113534)

[3.6.1 资源文件规范 15](#_Toc522113535)

[3.6.2 后台服务规范 15](#_Toc522113536)

[4 最佳实现及案例 17](#_Toc522113537)

[4.1 React代码部分 17](#_Toc522113538)

[4.2 Java代码部分 18](#_Toc522113539)

图 目 录

[图 1 数据多语字段示例 13](#_Toc521597806)

[图 2 后台资源加载定义 15](#_Toc521597807)

# 国际化架构简介

国际化的整体框架：采用定义标准的资源文件，然后通过组件进行加载、解析、展示的技术实现。

## 技术栈说明

基于IUAP应用平台以及React框架，保证功能性能满足要求的前提下，保持技术路线以及实现方式的一致，因此统一采用i18n插件的方式进行国际化集成。这样定义的资源标准可以进行统一，同时便于整理及维护。

React框架：采用React + React-intl的技术实现；

Java后端：统一采用java的resources的技术实现。

# 国际化工具使用说明

**国际化工具主要指的是针对前后端代码进行中文信息的抽取，然后对抽取的中文信息进行特定字符串替换的工具。**

基于现有已经存在平台以及业务应用系统，实现国际化的难度比较大，需要对所有现有的功能进行调整。为了使得调整的工作量尽量小，制作相应的辅助工具，辅助完成国际化的改造。

首先需要对所有的需要国际化的文本进行抽取，形成统一的资源文件，然后对在资源文件中的value值进行多语的翻译，最后再加载至系统中。该部分主要描述国际化工具的处理逻辑以及使用情况。

## 国际化工具处理逻辑

把需要国际化显示的信息抽取成多语资源文件，提供后台java代码、html代码、js代码等多语资源抽取的工具。把代码里的非注释信息的多语信息抽取为多语资源，自动按照多语资源文件规范生成多语资源文件。具体步骤如下：

1. 初始化项目目录

扫描工程下的所有目录，识别java、html、js等涉及中文的文件类型，同时形成文件的父子目录，用于最终的资源文件的目录生成。

1. 通过中文字符集范围进行文件抽取

通过正则匹配出所有的包含中文的内容，形成资源数据。

其次对找出的资源数据进行清洗，将不规则的数据变成规则的数据，去掉无法解析的资源数据等，形成最后的资源数据清单。

1. 将抽取的资源信息写入资源文件

将抽取的资源按照目录分类，生成资源的key值，然后以键值对的方式写入资源文件。

1. 将原始文件进行多语格式的替换

采用资源替换的方式对原始代码文件进行替换。该部分的解析难度比较大，需要考虑实际碰到的所有通用的以及特殊的情况。

1. 将抽取的资源以及替换后的文件放入系统工程进行调试

在抽取的过程中难免会碰到一些特殊的没有被处理的情况，因此抽取完成后，一方面需要将抽取的资源文件进行多语的翻译，一方面需要对更改后的文件进行测试，找出没有被自动化调整的地方并进行微调。确保代码的正确性。

## 国际化工具的配置

国际化工具已经上传至github，如有需要请下载至本地，按照说明使用。（URL：https://github.com/wenfan919/sany\_i18n）

国际化工具的各级目录说明如下：

sany\_i18n

├─jre （JAVA运行时环境）

│ ├─jdk1.8\*\*\*\* （尽量使用1.8以上的运行环境，需自行提供）

├─docs （国际化规范文档）

├─lib （国际化工具使用的jar包，主要是文件编码识别、日志等处理）

├─workbench （国际化的工作空间）

**├─config.properties （国际化配置文件）**

├─iuapi18n.log （国际化执行时的日志文件）

├─startup.bat （Windows下的执行脚本）

├─startup.sh （Linux下的执行脚本）

├─README.md （说明）

其中在使用国际化工具的过程中，主要的是针对配置文件的部分调整，说明如下：

1. parseProjectPath

该配置需要调整为自己的国际化工作空间（workbench）的完整路径

1. testMultiLangResourceType

该配置需要在使用工具生成json格式的资源文件或者properties格式的资源文件时进行调整。按照目前项目的情况，在生成java的国际化资源文件时采用properties格式；在生成React国际化资源文件时采用json格式。

示例：zh\_CN|properties,en\_US|properties

如果需要追加生成ja\_JP的properties资源文件，只需要在后面按照上面的格式追加即可。

1. replaceJSString

该配置主要是指将JS文件抽取的中文替换成的字符串的具体格式。在目前React框架下默认采用`<FormattedMessage id=\"{0}\" defaultMessage=\"{1}\" />`的方式，该部分暂时可以不用做调整。

## 国际化工具的使用

国际化的工具使用比较简单，具体步骤如下：

1. 按照上述的配置要求，将jre、config.properties文件进行调整；
2. 将需要国际化处理的文件放置在workbench中，尽量带上原始的工作目录；
3. 双击执行startup.bat文件，此时可以看到打开的窗口中出现滚屏的日志，当执行结束后窗口会自动关闭，即表示执行结束；
4. 抽取后生成的资源文件以及采用国际化工具替换后的代码统一存放在workbench下，请至该目录下查看；
5. 将workbench目录下的资源文件拷贝到项目的资源文件目录下，将该目录下的代码直接拷贝至原工程下对原代码进行覆盖即可。
6. 至此国际化工具使命结束，可以开始功能的调试了。

一、 java代码的工具使用

1、 修改config.properties文件中的testMultiLangResourceType的值为zh\_CN|properties,en\_US|properties

2、 将java代码拷贝到workbench目录下，要保证两级目录以上，最好是保证src之下的目录是完整的

3、 双击startup.bat

4、 将workbench目录下的刚才拷贝进去的文件夹直接覆盖原始文件即可（备注：其中对原始文件使用.temp备份，调试完成后可以删除）

二、 React前端代码的工具使用

1、 将React前端代码拷贝到workbench目录下，要保证两级目录以上，最好是保证src之下的目录是完整的

2、 双击startup.bat

3、 将workbench目录下的刚才拷贝进去的文件夹直接覆盖原始文件即可（备注：其中对原始文件使用.temp备份，调试完成后可以删除）

备注：

1. 如果需要查看日志，请打开文件iuapi18n.log进行查看；

# 国际化规范

国际化规范一方面从具体的前后端多语的实现上考虑，另一方面从多语的管理功能上考虑。其次关于历史的系统管理功能以及在此基础上构建的业务应用也纳入一并考虑并形成统一的规范要求。

国际化的具体特征如下，后续的设计规范也是基于此进行的整体编排：

1. 形成前后台资源规范

包含前后台多语资源文件规范。

包含前后台多语资源的路径规范。

最主要的是支持多语资源文件的扩展以及定制，第三方可以扩展自己需要的国际化资源文件。

1. 多语资源文件加载机制

确保资源文件在更新后，资源能进行实时加载，保证页面显示的正确性。

1. 多语种存取以及预制

支持中英文的多语资源预制。

1. 多语本地化处理

主要是系统功能以及业务功能，需要按照规范进行调整。

## 规范原则

在实施国际化的过程中，为了保证多语设计的完整性、高效性、稳定性等设计要求，需要定义一套国际化规范（主要包括前后端资源文件规范、路径规范、资源加载规范等），该规范应当遵循一定的设计原则，具体如下：

1. 易读性原则：

包名、key值等应当具有自表述能力，容易阅读和理解。

使用有意义的单词或缩写。

数字尽量使用有意义的常量代替。

良好的排版格式，特别是资源文件中key、value值前后不允许有空格。

1. 一致性原则：

在尽可能大的范围内，保持编码风格的一致性。

资源文件都必须是ISO-8859-1编码格式，对于所有非西方语系的处理，都要先转换为Unicode格式（转换方法是通过JDK自带的工具native2ascii）。

前后台资源编码规则等保持一致。

技术路线以及实现方式尽量一致。

资源文件内容的调整要保持所有的资源文件变更的一致性。

1. 最小内容原则：

在保证现有功能性能的前提下，系统功能及业务功能调整最小化。

尽量采用自动化工具实现国际化。

国际化的key尽量简单唯一，保证资源的请求效率。

## 统一规范要求

该部分的国际化支撑的统一规范要求对前后台同样适用。按照需求特性要求，需要考虑多语资源文件的命名规范以及内容的规范，同时考虑资源的加载机制以及替换机制。具体如下：

### 资源目录结构规范

资源文件按照功能模块的方式进行拆分。由于系统功能以及业务功能都是按照模块+功能+文件的方式进行划分，因此针对资源文件的存放也采用一致的目录结构，只是在最顶层目录不是src，而是locales。

具体请参见下面的目录结构，可见src、locales中除了叶子节点不一致外其他的目录结构都一致。

html以及js的资源文件放在一起。

java资源文件单独存放在/WEB-INF/i18n目录下，具体结构与locales类似。

webapp

├─locales

│ ├─default

│ └─app

│ ├─appList

│ ├─iuap.properties

│ ├─iuap\_zh.properties

│ ├─iuap\_en.properties

│ ├─appcenter

├─trd

├─src

│ ├─default

│ └─app

│ ├─appList

│ ├─index.html

│ ├─index.js

│ ├─index.css

│ ├─appcenter

├─vendor

├─index.js

├─index.html

├─README.md

### 资源文件命名规范

资源文件的命名规则为资源名称+“\_”+语言代码+“\_”+地域代码+“.properties”构成。

如果是默认的资源文件，则直接为资源名称+“.properties”构成，资源文件目录下都需要有一个默认的资源文件。

由于资源目录采用了与功能页面目录相同的设计，因此为了便于代码的统一书写以及代码混淆，资源名称统一采用小写字母“iuap”；

资源名称与语言代码及地域代码间使用下划线“\_”连接；

语言及地域编码采用ISO定义，具体请参见：https://github.com/wenfan919/sany\_i18n/tree/master/docs/语言及地域编码ISO.md

### 资源内容定义规范

资源内容采用单行：key=value的方式定义；

key值、value值的前后不允许有空格；

key值统一使用小写字母；

为了提高传输效率，减小资源文件加载过程的大小，在资源文件分目录的情况下，资源文件的key值采用简单的字母+序号的方式定义（组合后不大于4位）；

资源文件的key值的多单词间使用点“.”连接；

在采用国际化工具整理资源的情况下，基本的key命名规则是采用该文件的类型以及该文件所处层级的最后两个层级以及流水号组装而成；

示例如下：

|  |
| --- |
| js.app.use.0001=服务连接错误 |
| ht.app.use.0001=用户管理 |
| ht.app.use.0002=返回 |
| ht.app.use.0003=输入搜索内容 |
| ht.app.use.0004=高级搜索 |
| ht.app.use.0005=新增 |

如果自己想要自定义内容的key值，请尽量与自动生成的结构一致，并保证key不要重复。

### 其他国际化规范要求

针对时区等，所有日期和时间默认按照东8区时间，以long值方式存取。针对图片，支持图片资源国际化，对前端有关图片的多语资源进行统一管理，如公司logo图片，宣传图片等。

## 国际化管理规范要求

系统国际化的运转离不开语种以及多语数据的设置。语种的设置属于系统框架的定义范畴，多语的设置一方面属于平台自身的多语数据要求，另一方面属于基于平台开发的业务功能的多语数据要求。我们将一类归为多语资源预制，将另一类归为多语资源扩展。

### 语种的设置要求

首先系统管理员需要在系统管理界面设置系统的常用语种以及默认语种；

其次系统需要在登录页及“个人设置”页面定义用户自己的语种，当然默认语种为系统管理员设置的默认语种；用户自己修改的语种范围为常用语种；当用户修改完成后系统的展示便依据用户设置进行国际化调整。同时要保证当前用户的Locales的设置、Cookie的设置、缓存的设置、DB的多语属性保持联动一致。

### 多语资源预制要求

在语种描述表中预置系统可用的语种数据，默认预置简体中文、繁体中文、英文三种语种描述。

应用平台的多语资源也提供此三种语种的资源文件，用户在不进行任何设置的情况下可以完美使用此三种语种进行数据的展示等。当然针对二次开发的功能需要对多语资源进行追加调整，保证代码以及资源的一致性。

### 多语资源扩展要求

首先系统提供多语语种描述表数据模型，提供系统可用的语种存取API接口，开发人员可以通过该接口定义国际化语种以及或者所有语种信息，参数示例如下：

{"id":"", "prelocale":"zh\_CN","serialid":"","pageshow":"中文简体","description":"中文简体语种"}

另外，多语资源文件支持扩展和定制，第三方可以扩展自己需要的语种的国际化资源。按照本方案中定义的规范要求定义资源即可。

## 数据多语处理规范要求

数据多语部分主要包含两方面的内容：

一方面是多语数据的录入及存储，通过前端封装数据多语录入组件来实现对多语种数据信息的录入。另一方面是多语数据的展示，针对数据库中多语字段，需要支持多语种存储。因此需要对于这个字段进行扩展，目前按照项目要求最大支持六种语言。系统默认预置三种，分别是简体中文，繁体中文，英语。示例如下：

如user表中，对name字段进行扩展，添加name2,name3,name4,name5,name6这几个扩展字段。其中资源的命名规范为“原始字段名+序号”。后台提供标准的对象中字段获取的API接口，可以直接通过该接口获取多语字段的数据，示例如下：MethodUtils.getDataBySerial(wbAppMenu, "name", serial)。

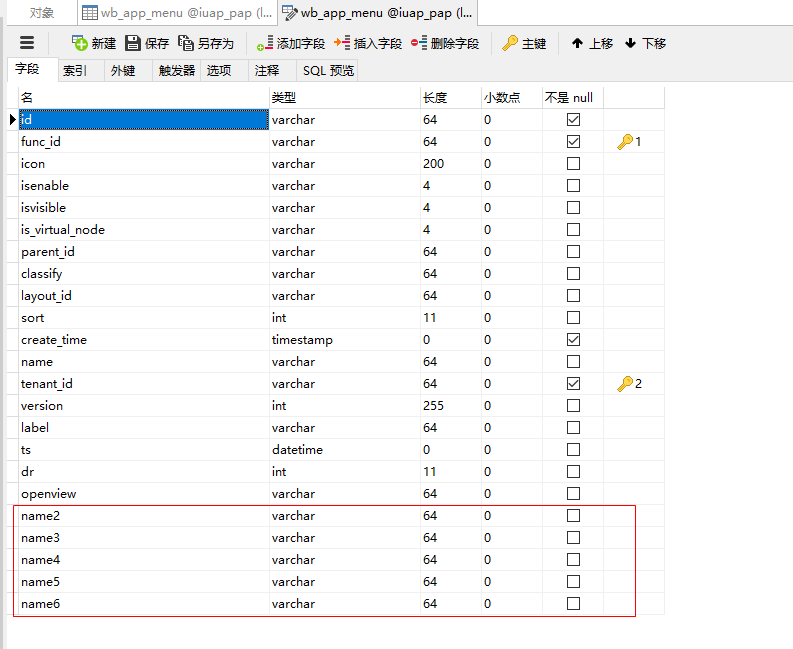


图 1 数据多语字段示例

## 基于React的国际化规范要求

React的国际化采用React-Intl组件进行集成。React-Intl是由yahoo开发的针对React的国际化类库，基于Format.js，支持语言、时间、货币等的国际化。

### React-Intl国际化步骤

1. 判断是否需要引入polyfill文件

如官方文档提到的那样，JavaScript有一套国际化标准API，React-Intl也是基于它的，但是由于Safari或者一些旧版本的浏览器不支持，于是我们需要在这些浏览器下引入polyfill文件来提升浏览器间的兼容性。

1. 引入 react-intl 的local data

react-intl在做国际化的时候需要一些特有的local data，主要是进行相对时间翻译时，比如昨天、今天、明天、几分钟前、几个月前之类的。我们通过addLocaleData这个方法加载相关内容，实际开发时可以根据实际情况加载需要的locale-data。

1. 创建react-intl国际化上下文组件

为了能够使用react-intl进行国际化，跟redux这些框架一样，我们需要一个Provider Component，用它来提供国际化的上下文。代码示例如下：

ReactDOM.render(

<IntlProvider locale={appLocale.locale} messages={appLocale.messages} formats={appLocale.formats} >

<Provider store={store}> <Routes history={window.appHistory} /> </Provider>

</IntlProvider>,

document.getElementById('\_\_react-content')

);

1. 使用 react-intl's components & apis，进行国际化开发

该部分正常引入react-intl组件，使用FormatedMessage() 即可进行资源的替换。

### 国际化资源文件管理

目前我们管理资源文件的方式是在工程根目录下的locales文件夹下。

\*.json是我们的资源文件(这里采用了 json 格式)，返回的是一个对象，key为我们翻译用的id，value为具体语言的翻译。其次en-US.js文件封装了messages、locale等国际化上下文组件需要的内容。有了这些资源文件以及相关的封装之后，我们就可以在InltProvider中使用了。

国际化资源文件自动化生成：该部分请直接参见第二章节。

## 基于后台服务的国际化规范要求

除了遵循国际化的统一规范要求之外，后端的国际化在资源文件的编制以及后端代码逻辑上也有不同的规范要求，具体如下：

### 资源文件规范

上面的资源文件中，属性值都是一般的字符串，它们不能结合运行时的动态参数构造出灵活的多语信息。要解决这个问题须使用带占位符的格式化串作为资源文件的属性值并结合使用MessageFormat就可以动态构造多语字符串。

示例：ja.app.wel.0001={0},您好，欢迎访问{1}系统！

### 后台服务规范

后台服务在加载多语资源的部分可以采用JDK提供的ResourceBundle，也可以采用Spring提供的ResourceBundleMessageSource。

首先在 applicationContext.xml 定义 messageSource:



图 2 后台资源加载定义

然后java的资源加载采用单例模式：MessageSourceUtil，在类加载时初始化资源文件的方式进行资源的统一加载。java应用代码中在使用资源时直接采用getMessage的方式进行调用即可。示例如下：

MessageSourceUtil.getMessage("ja.app.wel.0001",params[]);

# 最佳实现及案例

上述的规范要求不仅是针对平台的国际化规范要求，同时也针对基于平台的业务应用系统的国际化规范要求。实际应用开发人员在进行代码交付时遵循规范的同时可以参照最佳实践。

国际化工具主要是指对页面以及后台涉及的中文信息进行识别、抽取、替换等操作。由于页面的元素使用情况比较复杂，因此该工具在抽取以及替换的过程中可能存在一些问题，因此待国际化工具动作完成后需要按照规范做一些优化完善。

**最佳实践及案例的前提是通过国际化工具生成完资源以及文件替换，并将资源以及生成后的文件拷贝至指定目录下后的实践操作。**

## React代码部分

首先在通过国际化工具生成React前端资源时，需要确认国际化工具中config.properties配置文件中的配置项是否调整为了json格式的输出。即确保properties那一行被注释，json那一行被打开。

#testMultiLangResourceType=zh\_CN|properties,en\_US|properties

testMultiLangResourceType=zh\_CN|json,en\_US|json

1. 将资源文件拷贝到locales目录下之后，需要在intl组件中添加对资源文件的引用。打开/src/Intl/locales文件夹，可以看到下面有两个文件：en-US.js、zh-CN.js。打开这两个文件，分别添加资源文件的引用。

import CN from 'tinper-bee/locale/zh-cn.js';

import appLocaleData from 'react-intl/locale-data/zh';

import zhMessages from '../../../../locales/zh-CN.json';

import iuapZHMessages from '../../../../locales/iuap\_zh\_CN.json';

export default {

messages: {

...zhMessages,...iuapZHMessages,

},

tinperBee: CN,

locale: 'zh-CN',

data: appLocaleData,

};

其中标黄色的部分即为我们添加的资源文件的引用。

1. 对js文件中纯中文的字符（没有被引号括起来、没有被规则标签包含的字符等，该情况比较少见），需要自行添加相关资源到json资源文件（编号按照顺序累加），

"js.buy.com.0001":"中文字符"

同时使用下面的组件进行中文字符替换

<FormattedMessage id="js.buy.com.8888" defaultMessage="中文字符" />

1. 上面的两步如果没有问题了，就可以直接npm run 启动服务进行页面的调试了，按照目前情况，如果原始代码写的比较规范，基本不用做其他处理了，如果仍有问题请自行调整。
2. 针对可能出现的国际化替换的情况，除了第2步使用的组件的方案，React-intl同时提供API接口的实现方案，国际化工具已经在container.js文件中添加了对加载的组件的injectIntl的集成，在FormattedMessage组件使用有问题的情况下，可以使用this.props.intl.formatMessage({id:’’,defaultMessage:’’});API的方式进行调用，示例如下：

placeholder={**this**.props.intl.formatMessage({id:"js.Des.wor.0047", defaultMessage:"开始 ~ 结束"})}

## Java代码部分

首先在通过国际化工具生成Java后端资源时，需要确认国际化工具中config.properties配置文件中的配置项是否调整为了properties格式的输出。即确保properties那一行被打开，json那一行被注释。

testMultiLangResourceType=zh\_CN|properties,en\_US|properties

#testMultiLangResourceType=zh\_CN|json,en\_US|json

1. 首先在工程中的pom.xml文件中添加对i18n的依赖，具体如下：

<dependency>

<groupId>com.yonyou.iuap.baseservice</groupId>

<artifactId>iuap-pap-baseservice-i18n</artifactId>

<version>3.5.1-SNAPSHOT</version>

</dependency>

详细的pom文件可见：

http://maven.yonyoucloud.com/nexus/content/repositories/iUAP-Snapshots/com/yonyou/iuap/baseservice/iuap-pap-baseservice-i18n/maven-metadata.xml

1. 其次在工程中的web.xml中添加对request的监听。

<listener>  
 <listener-class>  
 org.springframework.web.context.request.RequestContextListener  
 </listener-class>  
</listener>

1. 再次在配置文件applicationContext.xml中添加应用上下文的工具类以及资源加载的定义。

<bean id="applicationContextUtil" class="com.yonyou.iuap.i18n.ApplicationContextUtil" lazy-init="false"></bean>

以及

<!-- 定义不同的日志、提示等信息的资源加载 -->  
<bean id="messageSource"  
 class="org.springframework.context.support.ReloadableResourceBundleMessageSource">  
 <property name="defaultEncoding" value="UTF-8" />  
 <property name="useCodeAsDefaultMessage" value="true" />  
 <!-- 生产上最好设置时间长一点，单位S -->  
 <property name="cacheSeconds" value="60"></property>  
 <!-- 将日志、提示信息分开 -->  
 <property name="basenames">  
 <list>  
 <value>/WEB-INF/i18n/iuap</value>  
 <!-- <value>/WEB-INF/i18n/logs</value> -->  
 </list>  
 </property>  
</bean>

1. 在功能目录/WEB-INF文件夹下添加i18n目录，并将资源文件iuap\_zh\_CN.properties、iuap\_en\_US.properties拷贝到该目录下。（确保资源文件的命名规范要求）