ISC微服务

后台框架开发规范说明书

版本：1.0

编制人：王孝坤

目录

[ISC微服务 1](#_Toc6405266)

[后台框架开发规范说明书 1](#_Toc6405267)

[一、 引言 3](#_Toc6405268)

[二、 后端获取前台用户登录信息 3](#_Toc6405269)

[1. 通过服务入参获取 3](#_Toc6405270)

[2. 通过线程变量获取 3](#_Toc6405271)

[3. 注意事项 4](#_Toc6405272)

[三、 包命名规范 4](#_Toc6405273)

[四、 接入层继承规范 5](#_Toc6405274)

[五、 DTO的扩展和自定义 5](#_Toc6405275)

[1. 默认出入参的扩展 5](#_Toc6405276)

[2. 自定义出入参 6](#_Toc6405277)

[六、 Mapper文件的扩展 6](#_Toc6405278)

[七、 通用方法的运用 7](#_Toc6405279)

[1. 精确查找 7](#_Toc6405280)

[2. 模糊查询 7](#_Toc6405281)

[3. 批量增删改 7](#_Toc6405282)

[4. 固定字段说明 7](#_Toc6405283)

[八、 数据库多语言实现方案 8](#_Toc6405284)

[九、 数据字典获取 9](#_Toc6405285)

[十、 消息码获取 10](#_Toc6405286)

[十一、 异常处理 10](#_Toc6405287)

[1. 系统异常 11](#_Toc6405288)

[2. 业务自定义异常 11](#_Toc6405289)

[十二、 数据版本控制 11](#_Toc6405290)

[十三、 版本修订历史记录 11](#_Toc6405291)

# 引言

本文是一套面向ISC微服务框架Java开发人员所应遵循的开发规范。基础Java程序的规范请参照《Java编码规范说明书》文档，本文档不再重述Java程序规范，而是基于使用ISC微服务后台框架上进行开发规范的说明，既是对框架说明的进一步补充，也可避免开发人员重复代码的编写

遵循此开发规范希望能达到以下目的：

* 对框架功能使用进一步了解
* 避免重复代码的编写
* 避免重复设计
* 减少开发工作量
* 统一通用场景的解决方案

# 后端获取前台用户登录信息

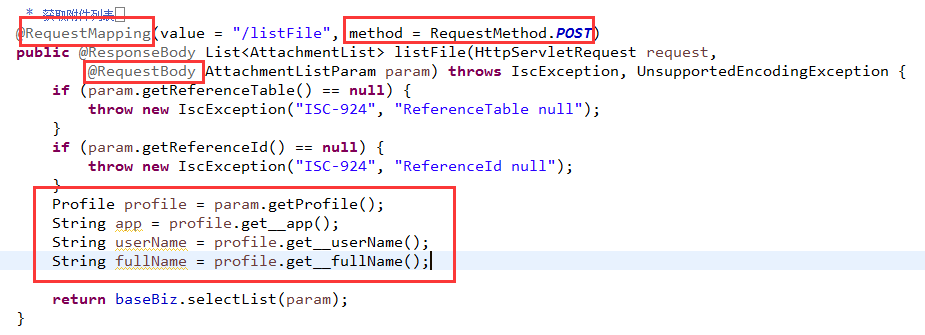
## 通过服务入参获取

通过入参获取登录用户信息需要符合以下两个条件：

1. 接入层为Controller包的Post请求，其他接入层Rpc或者Api不需要用户登录访问所以无法获取登录用户信息，方法添加RequestMapping且标记添加参数method = RequestMethod.POST或方法添加PostMapping标记都为Post请求。
2. 方法入参添加@RequestBody标记，且该入参继承自BasicParam基类

该类请求将被自定义消息转换器MyMappingJackson2HttpMessageConverter拦截并初始化入参，只有入参继承自BasicParam，才能够初始化BasciParam的Profile属性，Profile存储了用户的基本信息(用户Id/登录名/全名/国家/时区/当前语言/默认币种等)和应用扩展信息(令牌/当前应用/当前账套/事业部Id/组织Id/应用支持语言/应用扩展信息等)

如下图所示：

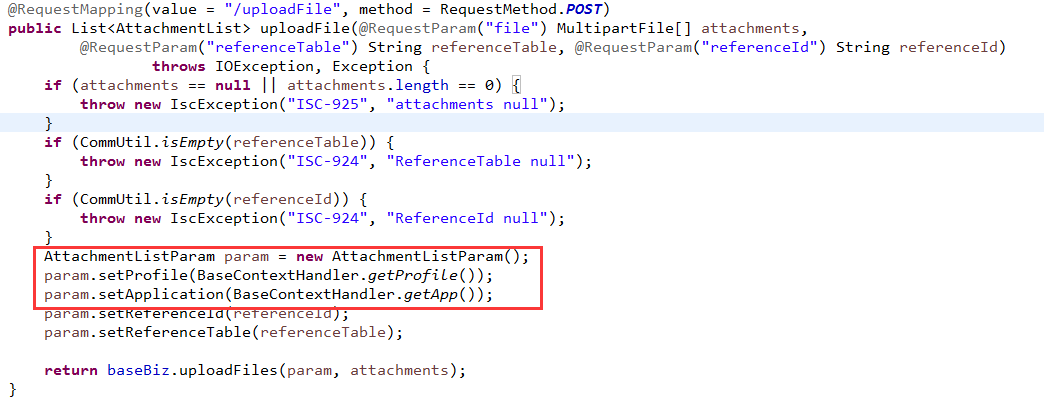


## 通过线程变量获取

只有来自接入层Controller的请求(理由如上)才能通过当前线程的本地变量获取登录用户的信息，ISC微服务框架内置的ServiceAuthRestInterceptor拦截器会在登录用户校验成功后设置BaseContextHandler类中的线程本地变量，目前提供以下方法获取用户信息：

|  |  |
| --- | --- |
| 方法名 | 说明 |
| getProfile() | 获取当前登录用户的所有信息 |
| getUserID() | 获取当前登录用户的Id |
| getUsername() | 获取当前登录用户的MIP账号 |
| getName() | 获取当前登录用户的全名 |
| getApp() | 获取当前登录的App名 |

使用方式如图所示：



应用场景说明：

1. 对于Post请求但入参不符合规范的请求
2. 对于非Post请求

## 注意事项

对于BasicRest和BasicApi的子类，不通过上述两种方式获取登录信息，因为在这两个基类下的服务不需要用户登录即可访问的，上述方式无法获取用户信息，如需要用户信息进行访问，请自行以入参方式传递。

# 包命名规范

包命名的统一前缀：com.midea.isc.project.module

作为微服务框架，一个项目可以分成多个服务模块，每个服务模块的工程是互相独立而业务上又是相关联的，固规定包名格式如上，例如销售项目(sale)划分仓储模块(wms)和订单模块(oms)则会独立生成两个项目工程，分别包名前缀为

com.midea.isc.sale.wms和com.midea.isc.sale.oms。

基于包名前缀再添加上述的约定包名形成具体包名(目前已约定的包名请参照ISC框架设计说明书)，如：

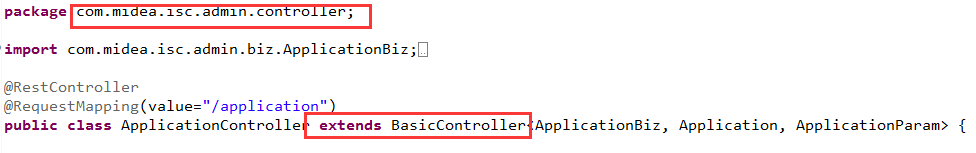
com.midea.isc.sale.wms.controller和com.midea.isc.sale.wms.rpc等等。

当需要添加不在约束范围内的包名时，只要保持前缀一致即可,如：

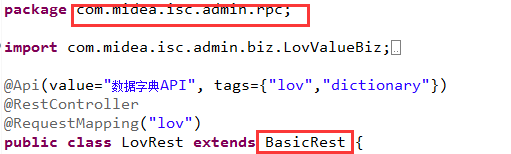
com.midea.isc.sale.wms.siebel。

# 接入层继承规范

controller包：提供前端页面的后台请求实现，需要用户登录才能访问，且由框架自动初始化入参的用户信息，服务间无法互调，继承自BasicController可以配套相应的框架通用操作，但不强制约束，如下图所示：



rpc包：提供服务间调用的请求实现，需要授权请求服务端后才能正常访问(服务调用方会在拦截器给请求头添加签名信息，Feign统一拦截器已经在auth-client包中注入)，swagger默认扫描的rpc包(可以添加swagger标记使接口说明更清晰)，所以需强制约束继承自BasicRest，如下图所示：



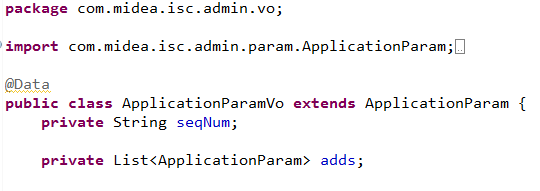
api包：提供外围系统调用的请求实现，需要指定网关发送的请求才能访问，做不同的安全策略控制，目前处于设计阶段未有应用场景校验

# DTO的扩展和自定义

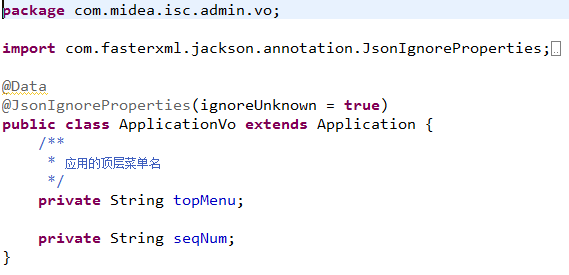
DTO是用于数据交换的传输媒介，对于原有DTO的扩展和自定义类，建议放到vo包中进行统一管理

## 默认出入参的扩展

框架默认的入参为\*\*\*Param，默认的出参为\*\*\*Model，有时基于业务处理，除了原有字段之外需要扩展入参添加用于逻辑判断的属性，或者扩展出参添加计算属性等等需求，此时建议入参通过添加子类继承对应的\*\*\*Param，出参通过添加子类继承对应的\*\*\*Model实现原有DTO的扩展，如图所示：



入参的扩展



出参的扩展

框架还提供了一个支持批量传递的入参BulkStruct，其中可以包括批量新增体、批量更新体、批量删除体，一般应用于父子关系实体中子表的行编辑保存，当需要扩展时例如添加父表信息传递，也可以通过继承它的方式实现。

默认出入参扩展类的命名建议以父类名称+Vo结尾。

## 自定义出入参

自定义入参：当某个业务逻辑处理不需要基于表结构进行入参定义和传递时，可以添加自定义入参类，如果为controller层使用的入参则可继承自BasicParam可以由框架初始登录用户信息，自定义入参类的命名以ParamVo结尾即可；

自定义出参：一般对于数据传输量有严格要求的移动端请求，框架提供的Model属性是完整版的，不符合实际需求，这时候可以自定义出参类，无需继承任何类，自定义出参类的命名以Vo结尾即可。

# Mapper文件的扩展

mapper.xml文件的扩展统一使用Ex结尾，例如Application-mapper.xml的扩展文件名为ApplicationEx-mapper.xml。对于扩展文件的namespace与原文件保持一致，使用同一个DAO接口，同时可复用原文件的sql语句段。

# 通用方法的运用

通用方法是指BaseBiz中提供的基础方法，本章节主要讲解如何灵活使用这些方法来进行业务逻辑的处理，同时使用正确的方式有助于程序运行性能的提高。

## 精确查找

精确查找建议使用selectOne单选和selectList集合两个方法，这两个方法是老框架保留方法，逻辑相对简单，所以myBatis对于SQL生成的效率高一点，精确查找不需要比较符的判断，使用精确匹配方式。

## 模糊查询

模糊查询使用find方法，新框架提供的新方法支持基于不同数据类型的数据列提供多种过滤方式，比较符说明和数据类型比较符映射关系请查阅《后台框架设计说明书》。这里要注意匹配符号的灵活运用可以减少代码工作量。

例如比较符me，可以实现基于字符串类型的in语句，这种对于值集的多选过滤提供快速的sql生成方式。

## 批量增删改

批量操作使用insertBulk、updateBulk、deleteBulk方法，批量方法的入参都是集合。注意：insertBulk暂不支持多语言表和Oracle的批量操作，updateBulk暂不支持多语言表的批量。

基于增删改的批量操作基础上增加saveBulk，可以在一个事务中实现批量新增修改删除，同时也提供便于前端调用的统一方式。

## 固定字段说明

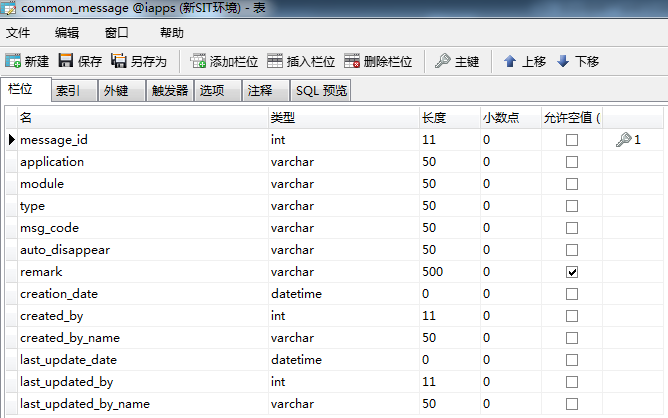
新表建议的固定字段createdBy、creationDate、lastUpdateBy、lastUpdateDate。BaseBiz提供setWho和setWhoForUpdateInfo，访问限制为protected，新增时使用setWho方法实现createdBy和lastUpdateBy字段的赋值，更新时使用setWhoForUpdateInfo实现lastUpdateBy字段的赋值。BaseBiz中提供的新增更新方法都已经自动调用上述两个方法。开发人员自己写的新增和更新方法，需要主动调用上述两个方法。注意只有入参继承自BasicParam且调用时存在用户登录信息时才能够实现自动赋值。

creationDate和lastUpdateDate建议使用sql方式进行设置(mysql使用now(),oracle使用sysdate)，主要是在mybatis生成sql语句时考虑，老框架是在代码中设置这两个字段的值，这种做法不合理，首先要考虑应用服务器和数据库服务器时区一致性问题，建议取时间统一使用数据库时间，其次lastUpdateDate可设计用于数据版本控制，用于代替乐观锁的版本号字段所以不能在应用中设置，而是要比较后才能取值。

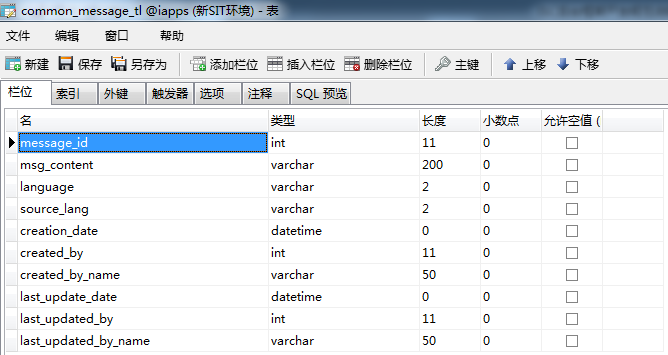
# 数据库多语言实现方案

ISC微服务框架提供的增删改查通用方法都已经支持基于数据库的多语言，如果想使用需要先了解框架的数据库多语言实现方案。

多语言方案如下，以消息码表为例：



业务主表



多语言从表

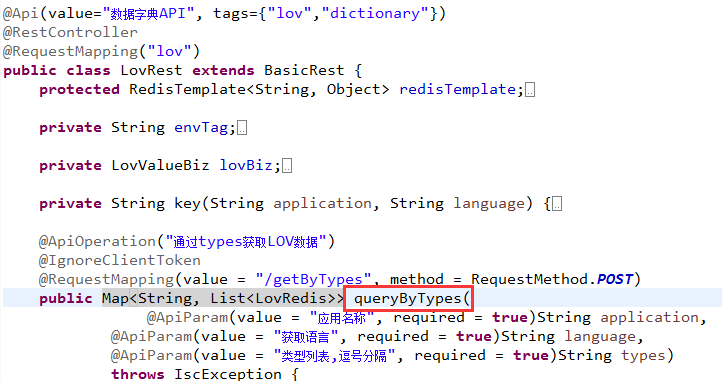
1. 业务表划分为两张表，多语言表以\_tl结尾，例如common\_message对应的多语言表为common\_message\_tl
2. 以业务表主键字段如message\_id作为相同记录的标志，多语言表以业务表主键字段message\_id+language作为唯一索引
3. 主表存放不需要多语言的字段，从表存放需要区分语言显示的字段，例子中的消息码表，只有msg\_content字段需要进行多语言存储
4. 多语言从表添加language和source\_lang，language标识该行记录的所属语言，source\_lang为记录新建时使用的语言。source\_lang的作用在于判断记录创建时的语言以及扩充语言时用于拷贝language=source\_lang的记录作为新语言的初始值。
5. 主表添加一行记录时，多语言表会根据当前应用所支持的语言例如支持zh,en,ja，生成3行记录，language分别为zh,en,ja，而soruce\_lang为当前登录用户所使用的语言，这时候虽然是三种语言，但其实存放的是同一种语言的值。只有在不同语言状态下更新后，才能看到不同语言的效果
6. 多语言方案要考虑一种应用场景，就是应用在使用过程中后续扩展了支持的语言，会导致多语言表原有记录缺少新语言记录。此时基于多语言的Update要考虑新语言记录的初始化问题，此时就需要步骤4中说明的source\_lang字段，要先拷贝language=source\_lang的旧记录作为新语言的初始记录，language为新语言的值，框架提供的方法为copySourceLang
7. 框架目前只支持单条记录增删改查的多语言方案，在代码生成器中对于多语言表重写了BaseBiz中的insert/update/updateByOtsLock等方法，同时更新主表和从表并添加了事务控制

# 数据字典获取

ISC微服务平台提供数据字典配置管理界面和获取服务，数据字典支持父子关系以及显示值的多语言。

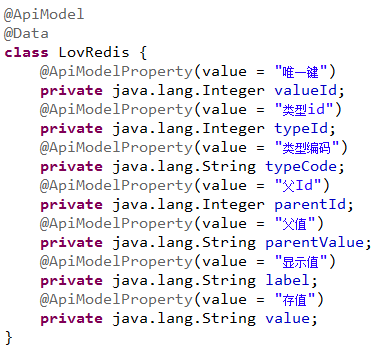
前端获取数据字典url为：/admin/lov/getByTypes

服务间获取数据字典的Feign接口定义为isc-admin的/lov/getByTypes



接口参数为应用名称app，获取语言language，类型types(多个type之间使用,分隔)。

返回值是以type作为键名的哈希Map，值为数据字典的集合，LovRedis定义如下：



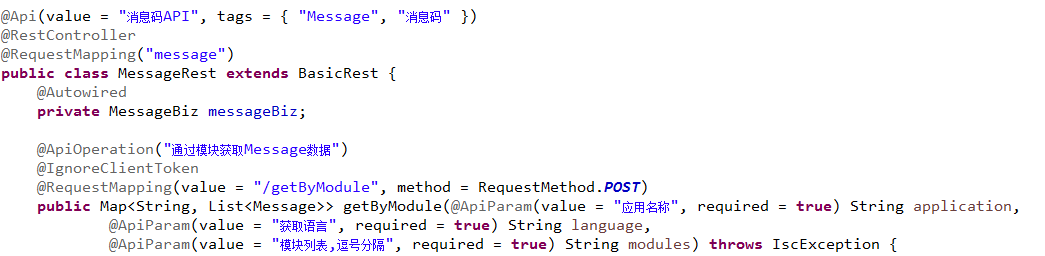
对于数据字典的获取服务为了提高性能后台已经进行了redis缓存处理，意味数据字典的配置修改需要更新缓存才能立马生效，配置界面已经提供了缓存刷新功能。缓存的设计是以lov+应用+语言+环境为单位分隔的哈希数据，例如lov:admin:zh:dev作为key名的哈希数据，field为type名，value为数据字典集合。

# 消息码获取

ISC微服务平台提供消息码配置管理界面和获取服务。

前端获取数据字典url为：/admin/message/getByModule

服务间获取数据字典的Feign接口定义为isc-admin的/message/getByModule



接口参数为应用名称app，获取语言language，模块列表modules (多个module之间使用,分隔)。

返回值是以module作为键名的哈希Map，值为消息码的集合。

消息码目前后台没有缓存设计，考虑后续添加。

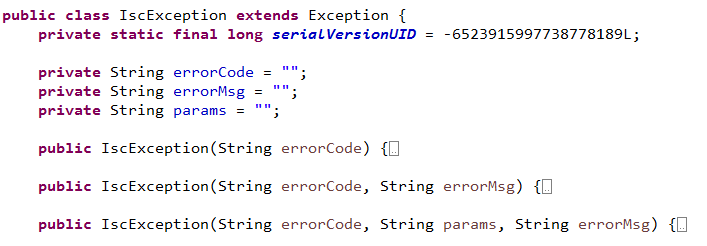
# 异常处理

## 系统异常

对于系统异常，ISC微服务框架已经进行了服务统一拦截和返回值封装，业务代码可以直接往上抛出来。系统异常统一返回的消息码为ISC-999，前端框架封装的http方法会自动识别后台系统异常进行统一提示，异常堆栈信息也会传递到前端通过主动查看的方式获取。

## 业务自定义异常

微服务框架对于后台的自定义错误需要中断业务处理过程直接返回的场景提供了自定义异常的统一封装，自定义异常类为IscException，并提供了多种构造函数，如图：



ErrorCode为消息码，例如ISC-001，ISC-002，INV-001，SO-001等

errorMsg为提示性消息，一般消息码的提示信息是通过后台配置获取的，此字段可用于开发人员存储用于定位问题的辅助信息

params为消息码对应的填充参数，例如消息码ISC-001的提示信息为Hello,s%,welcome to s%.params则设置为tom,Beijing。最终消息码显示为Hello,tom,welcome to Beijing.

自定义异常的使用方式如下：

简单方式：throw new IscException(“ISC-001”);

带提示：throw new IscException(“ISC-001”,”neet token!”);

带参数：throw new IscException(“ISC-001”,” tom,Beijing”,”neet token!”);

自定义异常也只需要往上抛，不用处理，ISC微服务框架已经进行了服务统一拦截和返回值封装

# 数据版本控制

数据更新时需要考虑数据版本是否过期，对于简单业务场景的单条数据更新建议使用框架提供的更新方法updateByOtsLock，该方法使用乐观锁的方案以最后更新时间字段lastUpdateDate作为版本号进行判断。更新成功lastUpdateDate也会设置为最新时间。

对于复杂的业务场景需要自己写sql更新数据库同时需要进行数据版本控制，框架只提供建议性方案，即使用lastUpdateDate作为版本判断字段，且更新成功获取数据库时间更新该字段。

# 版本修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 作者 | 内容 | 版本 |
| 2019-04-17 | 王孝坤 | ISC后台框架开发规范说明初版 | V1.0 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |