|  |  |
| --- | --- |
| 机密 | |
| **密级** | **一** |

**金证股份•J2EE开发平台**

**KJDP**

**(Kingdom J2EE Develop Platform)**

**服务端编程手册**

**[适用版本：3.5及以上版本]**

|  |  |  |  |
| --- | --- | --- | --- |
| **研制单位** | 深圳市金证科技股份有限公司 | | |
| **文档名称** | KJDP服务端编程手册 | | |
| **文档编号** |  | **文档版本** | 0.9 |
| **文档状态** | 🗹 草稿 🞏 完成 | | |
| **审 批** |  | **审批日期** |  |

**修改记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **文档版本** | **文档状态** | **修订日期** | **修订作者** | **修订内容** |
| 0.9 | C | 2014/5/5 | 林毅 | 新编制 |

文档状态标识：C –Created A – Added M – Modified D – Deleted

目录

# 引言

## 目的和范围

编写此指引旨在阐述**金证KJDP3平台**的后台设计思想、技术架构、开发管理、开发方法以及开发规范等方面内容，为使用此平台的开发项目组和技术人员提供指引，并规范开发及维护的过程。

此指引适用使用金证KJDP平台3.5及以上版本开发的业务系统以及衍生业务系统的开发过程及维护过程。

## 术语和定义

**KJDP**：金证J2EE开发平台(Kingdom J2EE Development Platform)

**Controller**：平台控制器入口层，接收http/https请求转化为内部请求调用内部服务完成处理

**Service**：平台服务层，结合服务配置工厂，完成服务的识别，提供内部、外部接口

**Atom**：原子业务，最小独立业务单元，完成独立、完整的业务操作

**Bex**：业务数据引擎，封装多数据源访问，通过统计接口提供到原子业务层、服务层调用

# 概述

## 设计思想

### 原子业务框架

原子业务在形式上是由多个互不相关的JAVA类组成，继承了统一的原子业务父类，通过统一的业务工厂类动态注册到服务器中供服务接口调用。

业务工厂类是是原子业务的框架，负责原子的动态装载及服务接口的调用，是基础框架层与原子业务类的纽带，串联原子业务为服务层调用提供支持。

数据缓冲区是原子业务框架的重要组成部分，为原子业务之间提供数据传递、数据共享的空间，解除了原子业务互相访问的耦合问题。数据缓冲区分为三个级别：全局数据缓冲区、服务数据缓冲区、业务数据缓冲区；三个数据缓冲区的生命周期各有不同。在多个服务中共享数据采用全局缓冲区；在服务内部的多个原子业务间共享数据采用服务数据缓冲区；在单个原子业务方法中采用业务缓冲区。

### 服务逻辑编排

服务中会包含一个或多个(Atom原子业务/Bex业务访问)配置，多个业务配置默认采用串行执行。在特殊场景下也可以支持业务逻辑的编排，业务逻辑编排使用groovy语言的语法，可进行分支if,循环for等逻辑判断完成实际业务场景。

<service service\_code="atomTest" service\_name="开户"

service\_type="1" service\_cate="1" arrange\_type="1" soap\_enable="1"

attr1="">

**<script><![CDATA[**

**def ret;**

**if ("$MODEL\_KEY" == "lytest")**

**{**

**ret = doCall("TestAtom2");**

**}**

**return ret;**

**]]></script>**

<businesses>

<service\_business business\_code="TestAtom1"

business\_name="TestAtom" business\_method="doUserQuery" serial\_no="1"

business\_type="3" business\_cate="1" attr1="" />

<service\_business business\_code="TestAtom2"

business\_name="TestAtom2" business\_method="doQuery2" serial\_no="1"

business\_type="3" business\_cate="1" attr1="" />

</businesses>

</service>

以上模拟代码中通过脚本编排，完成对入参的访问，判断服务入参MODEL\_KEY如果等于lytest那么服务调用会跳过第一个原子业务，而直接支持第二个原子业务。

## 逻辑结构

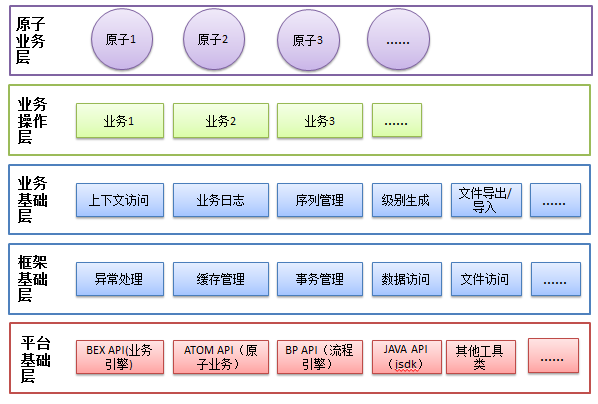


图 2-1 程序逻辑图

服务端程序按逻辑分五层：原子业务层、业务操作层、业务基础层、框架基础层、平台基础层

### 层次结构

* 平台基础层

由平台提供的API、开源工具类库、内部封装的类库等，以封装为类库的方式被其他模块进行调用，封装在AtomAbstract抽象类文件中。工具类库包括调用bex API、原子业务API、流程引擎API、JAVA API、第三方工具包等。

* 框架基础层

通过平台基础类派生的框架基础类BaseAtom类，继承平台基础层功能外，提供异常处理、缓存对象管理、业务间事务管理、数据访问、文件访问、异步处理等基础功能。

* 业务基础层

通过框架基础类派生的业务基础类BizAtom类，继承上层功能外，提供权限控制、业务日志、序列管理、级别生成、文件导入/导出等功能支持。

### 调用规范

服务端编程的各层对象设计时需遵循以下调用规范：

1. 原子业务层不能直接进行数据访问，需要通过业务操作层提供的方法封装进行访问。
2. 业务模型层设计需要与实体对象一一对应，业务操作层尽量安装业务模型进行划分，如客户联系资料模块就会有一个联系资料的模型类，一个联系资料的业务操作类，通过原子业务包装提供到服务层调用。
3. 原子业务方法间调用需要通过框架基础层提供的统一访问接口进行访问，不可直接互相调用。
4. 原子业务层、业务操作层不可直接调用平台基础层接口，需要通过框架基础层/业务基础层封装的接口间接访问。

## 核心类图



图2-2 核心类图

如上图所示，AtomAbstract抽象类是所有原子业务类的基类，BaseAtom继承AtomAbstract实现框架基础层支持,BizAtom继承BaseAtom实现业务框架层支持，组成原子业务核心模型：模型+操作，即原子业务调用业务操作类产生业务IModel对象。

# 平台基础层

## Bex API（业务引擎接口）

Bex主要通过BexBusinessImpl类完成业务引擎访问，数据访问的核心入口函数doBusiness，函数原型：public GenericResult doBusiness(GenericRequest request);

函数参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| request | 统一请求对象 | Object/ GenericRequest |

函数返回：返回统一结果集GenericResult 内部结构包（flag、prompt、rows、times、dataList）等元素。

函数说明：发起bex数据请求，根据bex配置完成对不同数据源的访问，会根据bexType进行数据源对象选择，对应执行不同数据源的数据访问方法，以下是不同数据源对应类型

/\*\*

\* **@1**-SP\_BEX 存储过程接口

\* **@2**-LBM\_BEX KCBP LBM 接口

\* **@3**-BEAN\_BEX JavaBean 接口

\* **@4**-BATIS\_BEX ibatis/mybatis 接口

\* **@5**-SQL\_BEX SQL 数据集接口

\* **@6**-WEBSRV\_BEX Web Service 接口

\* **@8**-KCXP\_BEX kcxp 中登网关接口

\* **@9**-KESB\_BEX 老lbm接口

\* **@10**-KCXP\_BEX kcxp 银行网关接口

\* **@11**-META\_SP\_BEX 针对元数据处理的存储过程

\* **@13**-ESB\_WEBSRV\_BEX ESB的webservice接口

\* **@14**-ESB\_WEBSRV\_BEX ESB的发布接口

\*/

## Atom API（原子业务接口）

## BP API（流程引擎接口）

流程引擎接口通过ProcessBiz类进行封装，通过dispatchBpmRequest方法来调用引擎的代码完成流程相关操作；

函数原型：public GenericServiceResult dispatchBpmRequest(GenericServiceResult genericServiceResult,Map requestParams,Map commParams)

函数参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| genericServiceResult | 业务结果对象 | Object/ GenericServiceResult |
| requestParams | 请求参数集合类 | Object/Map |
| commParams | 公共参数集合类 | Object/Map |

函数返回：返回服务结果集GenericServiceResult内部包含服务请求结果和流程引擎请求结果。

函数说明：

请求会根据requestParams请求参数集合中BPM\_OP决定调用流程引擎哪个方法，具体方法调用参见附件【流程通用接口使用说明】。



## LogUtil（日志工具）

LogUtil日志工具类，实现一些共用操作，通过静态方法的封装完成对日志的方便调用。分别提供不同级别的错误记录函数

|  |  |  |
| --- | --- | --- |
| 级别 | 函数 | 说明 |
| debug | LogUtil. debug() | 对应log4j调试级别，在开发中关键程序及逻辑中会加入 |
| info | LogUtil. info() | 对应log4j提示级别，在用户常规信息显示 |
| warn | LogUtil. warn() | 对应log4j警告级别，在程序可能出现问题或某些特殊处理会使用警告级别，提示管理员检查系统 |
| error | LogUtil. error() | 对应log4j错误级别，在程序出错是会提示错误级别日志 |

## CosUtil文件管理工具

### 文件上传（upload）

函数原型：public static String upload(HttpServletRequest request,IService service,Map<String, String> commParams) throws IOException;

函数参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| request | 统一请求对象 | Object/ GenericRequest |
| service | 服务调用类 | Object/ IService |
| commParams | 公共参数 | Object/ Map |

函数返回：返回字符串 返回服务器端文件信息json串，内部结构包（att\_id/filename/filecon/filesize）等元素。

函数说明：upload方法将request中文件数据流转换成文件流写入服务器端文件，并产生随机文件名，记录到数据库附件表并返回文件信息。

### 文件删除（delete）

函数原型：public static String delete(HttpServletRequest request, IService service,Map<String, String> commParams) throws IOException;

函数参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| request | 统一请求对象 | Object/ GenericRequest |
| service | 服务调用类 | Object/ IService |
| commParams | 公共参数 | Object/ Map |

函数返回：返回字符串 返回服务器端文件信息json串，内部结构包（att\_id/filename/filecon/filesize）等元素。

函数说明：upload方法将request中文件数据流转换成文件流写入服务器端文件，并产生随机文件名，记录到数据库附件表并返回文件信息。

### 文件下载（download）

函数原型：public static File download(HttpServletRequest request, IService service) throws IOException;

函数参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| request | 统一请求对象 | Object/ GenericRequest |
| service | 服务调用类 | Object/ IService |
| commParams | 公共参数 | Object/ Map |

函数返回：返回字符串 返回服务器端文件信息json串，内部结构包（att\_id/filename/filecon/filesize）等元素。

函数说明：upload方法将request中文件数据流转换成文件流写入服务器端文件，并产生随机文件名，记录到数据库附件表并返回文件信息。

## DateUtil日期处理工具

### convertStringToDate(字符串转日期)

### covertDateToString(日期转字符串)

### covertLongToString(长整形转字符串)

## DomUtil文档处理工具

### docToFile(文档转文件)

### resourceToDoc(资源对象转文档)

### stringToDoc(字符串转文档)

## JsonUtil JSON对象处理工具

### json2Map(json转集合)

### json2MapTotally(json转集合全递归)

### json2Xml(json转xml)

### map2Json(集合转json)

### map2Xml(集合转xml)

### req2Json(请求转json)

### str2XmlNode(字符串转xml)

### string2Json(字符串转json)

## PoiUtil 文件导出处理工具

### exportFile(文件导出)

### parseImportFile(文件导入)

### transCharacterEncoding(字符集转换)

重发错误码与撤销错误码不允许有重复，前端维护功能需要控制。

# 异常处理

### 异常处理流程

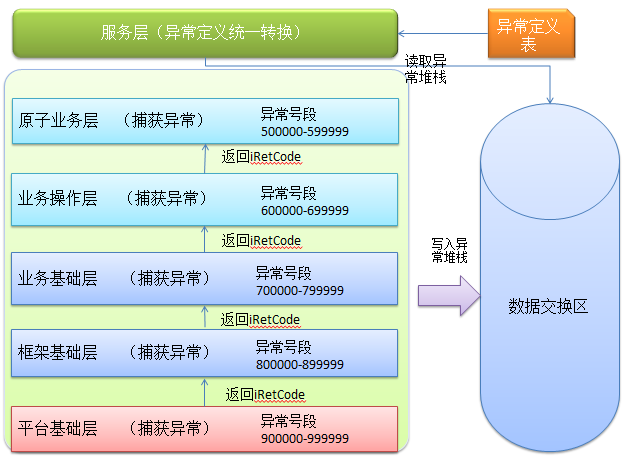


图4-2异常处置流程

程序中异常捕获

### 异常相关表

* 系统错误表UPM\_SYS\_ERROR

|  |  |  |
| --- | --- | --- |
| **字段名** | **字段说明** | **数据类型** |
| ERR\_CODE | 错误编号 | VARCHAR2(32) |
| ERR\_MSG | 错误消息 | VARCHAR2(256) |
| ERR\_LVL | 错误级别 | NUMBER(2) |

### 异常堆栈



图4-3异常堆栈结构

异常堆栈内部存储使用Map集合进行存储，异常捕获后需要执行

异常堆栈调用案例

ErrorStack es = new ErrorStack("800000","【框架基础层异常】:"+JsonUtil.string2Json(genericResult.getPrompt()),FrameworkConstants.FRAME\_BASICE\_LVL); DataExchangeAware.addExceptionStack(FrameworkConstants.FRAME\_BASICE\_LVL, es);

throw new AtomException(es);

# 业务基础层

业务基础层包：com.szkingdom.frame.business.atom.bizcomm

## 序列发生器（KUI\_SEQNEXTVAL）

功能描述：用于统一序列字段产生方法，兼容不同数据库的序列代码统一处理。

函数入参：Map[SEQ\_NAME]序列名

函数出参：LVL

## 级别创建器（KUI\_CREATELVL）

功能描述：用户统一创建级别方法，兼容不同数据库的级别字段统一产生。

函数入参：Map[TB\_NAME,ID\_FLD,PID\_FLD,LVL\_FLD,PID,LVL\_LEN]表名,主键字段,父节点字段,级别字段,父节点,级别长度。

函数出参：ID

函数代码：

public static String KUI\_CREATELVL(Map<String, String> params) {

try {

String lvlFld = params.get("LVL\_FLD");

String tbName = params.get("TB\_NAME");

String pidFld = params.get("PID\_FLD");

String idFld = params.get("ID\_FLD");

String pid = params.get("PID");

if (lvlFld == null || tbName == null || pidFld == null

|| pid == null || idFld == null) {

return null;

}

GenericResult genericResult = FrameDao.doBexCall(

"frameParMaxLvlQuery", params, null);

String newLvl = "0000000000000000000000";

int lvlLen = Integer.parseInt(params.get("LVL\_LEN"));

if (genericResult.getDataList() != null

&& genericResult.getDataList().size() > 0

&& genericResult.getDataList().get(0) != null) {

String maxLvl = (String) genericResult.getDataList().get(0)

.get("MAX\_LVL");

String pLvl = (String) genericResult.getDataList().get(0)

.get("PAR\_LVL");

if(pLvl==null)pLvl="";

if (maxLvl != null) {

int iMax = Integer.parseInt(maxLvl.substring(

maxLvl.length() - lvlLen, maxLvl.length()));

newLvl = newLvl + (++iMax);

} else {

newLvl = newLvl + "1";

}

return pLvl

+ newLvl.substring(newLvl.length() - lvlLen,

newLvl.length());

} else {

newLvl = newLvl + "1";

return newLvl.substring(newLvl.length() - lvlLen,

newLvl.length());

}

} catch (AtomException exception) {

ErrorStack es = new ErrorStack("700010",

"【业务基础层异常】:KUI\_CREATELVL级别计算函数执行失败",

FrameworkConstants.BIZ\_BASICE\_LVL);

DataExchangeAware.addExceptionStack(

FrameworkConstants.BIZ\_BASICE\_LVL, es);

throw new AtomException(es);

}

}

# 业务操作层

## 业务操作类

### 设计规范

1. 后台表结构只能由唯一业务操作类进行DAO操作。
2. 一个业务操作类可以操作一个或多个相关的表。

### 开发规范

1. 包名规范 com.szkingdom.business.atom.+模块名+.biz

例如：com.szkingdom.business.atom.param.biz 为参数管理模块业务操作类的包

1. 类名规范 功能名+Biz

例如：SysMenuBiz

1. 方法名规范

根据不同的操作操作方法分以下命名：

新增操作（insert+功能名）

修改操作（update+功能名）

查询操作（select+功能名）

删除操作（delete+功能名）

获取操作（get+功能名）

批量新增（batch+功能名）

### 代码案例

public String getSysMenuLvl(String pid) {

try {

Map<String, String> lvlParam = new HashMap<String, String>();

lvlParam.put("LVL\_FLD", "MENU\_LVL");

lvlParam.put("TB\_NAME", "UPM\_MENU");

lvlParam.put("PID\_FLD", "PAR\_MENU");

lvlParam.put("ID\_FLD", "MENU\_ID");

lvlParam.put("PID", pid);

lvlParam.put("LVL\_LEN", "4");

String lvl = BizCommAtom.KUI\_CREATELVL(lvlParam);

return lvl;

} catch (AtomException exception) {

ErrorStack es = new ErrorStack("600011",

"【业务操作层异常】:getSysMenuLvl获取菜单级别数据操作失败",

FrameworkConstants.BIZ\_LVL);

DataExchangeAware.addExceptionStack(FrameworkConstants.BIZ\_LVL, es);

throw new AtomException(es);

}

}

# 原子业务层

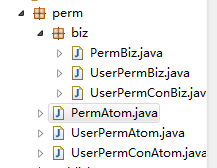
## 原子业务类

### 设计规范

### 开发规范

1统一继承AtomAbstract

2在com.szkingdom.business.atom包下进行开发

3 类分为原子业务类和操作类

### 代码案例

package com.szkingdom.business.atom.perm;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.apache.commons.lang.StringUtils;

import com.szkingdom.business.atom.perm.biz.PermBiz;

import com.szkingdom.frame.business.atom.AtomResult;

import com.szkingdom.frame.business.atom.basic.AtomAbstract;

import com.szkingdom.frame.business.atom.bizcomm.BizCommAtom;

import com.szkingdom.frame.business.atom.exchange.DataExchangeAssembly;

import com.szkingdom.frame.config.FrameErrorCode;

import com.szkingdom.frame.config.FrameworkConstants;

import com.szkingdom.frame.exception.AtomException;

import com.szkingdom.frame.service.model.GenericResult;

public class PermAtom extends AtomAbstract {

private PermBiz permBiz = new PermBiz();

@SuppressWarnings("unchecked")

public GenericResult executeDelPerm(DataExchangeAssembly dataExchange) {

try {

Map<String, String> params = (Map<String, String>) dataExchange

.getBusinessData("params");

Map<String, String> commParams = (Map<String, String>) dataExchange

.getBusinessData("commParams");

int iRetCode;

String[] permPur = new String[1];

iRetCode = permBiz.selectPurByMID(params, commParams,permPur);

if(iRetCode!=FrameErrorCode.ISUCCESS\_CODE){

throw new AtomException(-500011,

"【原子业务层异常】:executeDelPerm得到权限菜单用途权限原子业务执行失败",

FrameworkConstants.ATOM\_LVL);

}

if(("1").equals(permPur[0])){

params.put("PUR", "1");

}else{

params.put("PUR", "2");

}

String[] permLvl = new String[1];

iRetCode = permBiz.selectLvlByMID(params, commParams,permLvl);

if(iRetCode!=FrameErrorCode.ISUCCESS\_CODE){

throw new AtomException(-500011,

"【原子业务层异常】:executeDelPerm得到权限菜单级别权限原子业务执行失败",

FrameworkConstants.ATOM\_LVL);

}

if(StringUtils.isEmpty(permLvl[0])){

throw new AtomException(-500011,

"【原子业务层异常】:executeDelPerm得到权限菜单级别权限原子业务执行失败，没有关联权限数据",

FrameworkConstants.ATOM\_LVL);

}

params.put("PERM\_LVL", permLvl[0]);

iRetCode = permBiz.delSysPerm(params, commParams);

if(iRetCode!=FrameErrorCode.ISUCCESS\_CODE){

throw new AtomException(-500011,

"【原子业务层异常】:executeDelPerm删除权限原子业务执行失败",

FrameworkConstants.ATOM\_LVL);

}

return new AtomResult("0", "权限删除成功!",null);

}catch (AtomException exception) {

return new AtomResult(exception.getErrorCode(),exception.getMessage());

}

}

}

## 原子业务配置

### 配置要求

### 配置案例

<atoms>

<atom atom\_code="insertSysMenu" atom\_name="新增系统菜单" atom\_class="com.szkingdom.business.atom.param.SysMenuAtom" atom\_method="executeInsertMenu" attr1="" attr2=""/>

<atom atom\_code="insertSysPerm" atom\_name="新增系统权限" atom\_class="com.szkingdom.business.atom.perm.PermAtom" atom\_method="executeInsertPerm" attr1="SEQ\_PERMISSION" attr2=""/>

<atom atom\_code="insertrelPlat" atom\_name="新增系统关联平台" atom\_class="com.szkingdom.business.atom.param.RelPlatAtom" atom\_method="executeInsertRelPlat" attr1="SEQ\_UUM\_PLATFORM" attr2=""/>

</atoms>

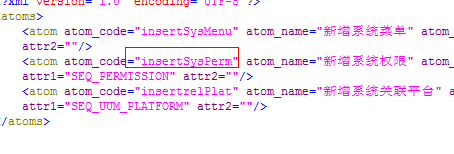
## 服务配置

### 配置要求

1、业务代码配置

business\_code="insertSysPerm"

配置业务代码为原子业务配置文件中原子业务代码



2、业务类型配置

business\_type=”3”

配置原子业务需要使用表示业务类型为原子业务

### 配置案例

<service\_business business\_code="insertSysPerm" business\_name="新增系统权限原子业务" business\_method="" serial\_no="1" business\_type="3" business\_cate="1" attr1="" />