金蝶云.苍穹定制化开发规范

苍穹平台解决方案部

文档简要信息：	
文档主题(Title)	苍穹定制化开发规范
作者(Author)	苍穹平台解决方案部
审批者 (To Be Approved By)	袁洋
说明 (Comments)	
文件名称(File Name)	

文档版本历史：

序号	日期	变更说明	修改人	注释
	2021-03-11	合并	苏绮静	合并历史版本，形成统一基础版本
	2021-03-18	修改	苏绮静	（1）补充命名规范内容补充 （2）补充部分示例 （3）格式调整
	2021-04-06	修改	苏绮静	整合《金蝶云.苍穹定制化开发_平台规范 V2.1》 （1）更新命名、设计规范 （2）新增多语言、微服务开发
	2021-04-16	修改	廖继红	请张利军，陈丰，魏向阳，郑政芳评审
	2021-05-13	修改	廖继红	更新排版，增加推荐，强制分类。
	2021-05-27	修改	廖继红	苍穹国际产品部汪东海，谭欣，王宁补充多语言规范。
	2021-07-29	修改	苏绮静	更正 2.4 中的第 3 点，删除第 4 点；更正 7 异常规范的实例化异常；新增 8 日志规范第 6 点
	2021-09-24	修改	苏绮静	删除报表相关模块，新增安全开发规范内容

目录

1 引言

1.1 编写目的

为保证苍穹系统稳定、高效运行，规范苍穹定制化设计与开发，促进业务健康发展，根据公司相关规定，结合当前多个项目运行系统的实际情况，特制定苍穹定制化开发规范。

1.2 使用范围

苍穹平台定制化开发

1.3 名词解释

1. Pascal 大小写：大小写形式 - 所有单词第一个字母大写，其他字母小写。例如：BackColor。
2. Camel 大小写：大小写形式 - 除了第一个单词，所有单词第一个字母大写，其他字母小写。例如：backColor。
3. <label>：语法块的名称。此约定用于对可在语句中的多个位置使用的过长语法段或语法单元进行分组和标记。
4. { }（大括号）：必选语法项。
5. [ ]（方括号）：可选语法项。
6. | （竖线）：分隔方括号或大括号中的语法项。只能使用其中一项。
7. 命名空间：命名空间提供了一种组织相关类和其它类的方式。命名空间是一种逻辑组合，既用作程序的“内部”组织体系，也用作“外部”组织体系（一种表示向其他程序公开程序元素的途径）。命名空间和程序集有助于开发基于组件的系统。

8. 程序集：用于物理打包和部署。程序集可以包含类型、用于实现这些类型的可执行代码以及对其他程序集的引用。

9. KSQL：金蝶数据库结构化查询语言。SQL92 的子集。

10. 健康中心：新产品系统性能、错误诊断框架，继承自 EAS 健康中心。

11. 插件：基于平台插件，通过平台提供的接口，只需要实现各个功能特殊的维护逻辑、叙事簿显示取数等接口，通过配置动态增加业务功能。

12. 扩展函数：平台高级扩展，针对业务系统需求扩展公共服务功能，如：取当前日期、按表达式计算、字段携带的逻辑运算等。

13. 产品：即某业务开发云。

1.4 参考资料

《苍穹平台规范 V1.1》

《金蝶云.苍穹定制化开发_平台规范 V2.1》

《安全开发不能不知道的 8 大防范措施》

《金蝶云苍穹多语言支持开发规范》

2 命名规范

2.1 扩展项目命名规范

【推荐】书写格式：<isv 标识>-<产品云代号>-<系统代号>[-工程类型]-ext

说明：

isv 标识：开发商标识；

产品云代号：产品的开发代号（如：bos、fi、hr）；

系统代号：按系统[技术框架 | 应用编码]划分代号（如：ksql、gl、ep、sal）；

工程类型：如表单插件工程(formplugin)、操作插件工程(opplugin)、报表工程（report）等。

如果是自主新增的云、应用，不属于扩展，不需要用到 ext。

正例：

isv-fi-gl-formplugin-ext（isv 为开发商标识）

2.2 包命名规范（扩展）

【强制】书写格式： <开发商标识>.<产品云代号>[.应用编码][.功能][.后缀]

说明：

开发商标识：开发商标识；

产品云代号： 产品的开发代号（例如：bos、fi、hr）；

功能：模块的功能名，可以增加子功能 [.子功能]；

后缀：.{test | plugin }

正例：

插件包命名：isv.fi.gl.formplugin

财务总账包命名：isv.fi.gl.voucher(其中 isv 为开发商标识)

2.3 类/插件命名规范

2.3.1 常规类命名规范

1. **【推荐】类名使用 UpperCamelCase 风格。**

   说明：但以下情形例外：DO/BO/DTO/VO/AO/PO/UID 等 。

   正例：

   JavaServerlessPlatform/UserDO /XmlService/TcpUdpDeal/TaPromotion

   反例：

   javaserverlessplatform/UserDo/XMLService/TCPUDPDeal/TAPromotion

2. **【推荐】杜绝完全不规范的缩写，避免望文不知义。**

   反例：

   AbstractClass 缩写成：AbsClass；condition 缩写成：condi；

   注：此类随意缩写严重降低了代码的可阅读性。

3. **【推荐】抽象类命名使用 Abstract 开头。**
4. **【推荐】异常类命名使用 Exception 结尾。**
5. **【推荐】测试类命名以它要测试的类的名称开始，以 Test 结尾。**
6. **【推荐】接口和实现类的命名有两套规则：**

（1）对于 Service 和 DAO 类，基于 SOA 的理念，暴露出来的服务一定是接口，内部的实现类用 Impl 的后缀与接口区别

正例： CacheServiceImpl 实现 CacheService 接口；

（2）如果是形容能力的接口名称，取对应的形容词为接口名（通常是－able 的形容词）

正例： AbstractTranslator 实现 Translatable 接口。

7.**【推荐】枚举类名带 Enum 后缀，枚举成员名称需全大写，单词间用下划线隔开。**

说明：枚举其实就是特殊的类，域成员均为常量，且构造方法被默认强制是私有；

正例： 枚举名字为 ProcessStatusEnum 的成员名称：SUCCESS / UNKNOWN_REASON。

8.**【推荐】各层命名规约：**

Service/DAO 层方法命名规约：

（1）获取单个对象的方法用 get 做前缀；

（2）获取多个对象的方法用 list 做前缀，复数形式结尾如：listObjects；

（3）获取统计值的方法用 count 做前缀；

（4）插入的方法用 save/insert 做前缀；

（5）删除的方法用 remove/delete 做前缀；

（6）修改的方法用 update 做前缀。

领域模型命名规约数据对象：

（1）xxxDO，xxx 即为数据表名；

（2）数据传输对象：xxxDTO，xxx 为业务领域相关的名称；

（3）展示对象：xxxVO，xxx 一般为网页名称；

（4）POJO 是 DO/DTO/BO/VO 的统称，禁止命名成 xxxPOJO。

2.3.2 插件命名规范

1. **【推荐】按模块分工程，每个模块分为：表单插件（含表单、基础资料、单据、列表）、操作插件、报表插件。**

   正例：

   （1）表单插件：{插件名}Plugin 如：FormPlugin；

   （2）单据插件：{插件名}Edit 如：PurchaserEditPlugin；

   （3）列表插件：{插件名}List 如：PurchaserListPlugin；

   （4）操作插件：{插件名}Op 如：PurchaserOpPlugin；

   （5）报表插件：{插件名}Rpt 如：PurchaserRptPlugin。

2. **【推荐】插件继承：**

	类（插件、服务）	继承自抽象类
平台插件	单据插件	AbstractBillPlugin
	基础资料插件	AbstractBasePlugIn
	动态表单插件	AbstractFormPlugIn
	列表插件	AbstractListPlugIn
	树型基础资料列表	StandardTreeListPlugin
移动插件	移动表单插件	AbstractMobFormPlugin
	移动单据插件	AbstractMobBillPlugin
	移动列表插件	AbstractMobListPlugIn

操作服务	操作服务插件	AbstractOperationServicePlugIn
单据转换	单据转换插件	AbstractConvertPlugIn

2.4 方法命名规范

1. **【推荐】使用 lowerCamelCase 风格，必须遵从驼峰形式**

正例： localValue / getHttpMessage() / inputUserId。

2. **【推荐】简单明了：**

根据上下文给动词和介词加上名词。请使用 removeObject(object, atIndex: index)，而不是 remove(object, at: index)。不要为了过度的简洁而影响清晰准确性。

3. **【推荐】避免缩写：**

使用 printError(myError) 而不是 printErr(myErr)，以及 setBackgroundImage(myImage)而不是 setBGImage(myImg)。

2.5 变量命名规范

1. **【推荐】在常量与变量的命名时，表示类型的名词放词尾，以提升辨识度。**

正例： startTime / workQueue / nameList / TERMINATED_THREAD_COUNT;

反例： startedAt / QueueOfWork / listName / COUNT_TERMINATED_THREAD。

2. **【推荐】避免在子父类成员变量之间、或不同代码块的局部变量之间采用完全相同的命名，使可读性降低。**

说明：子类、父类成员变量名相同，即使是 public 类型的变量也是能够通过编译，而局部变量在同一方法内的不同代码块中同名也是合法的，但是要避免使用，对于非 setter/getter 的参数名称也要避免与成员变量名称相同。

3. **【推荐】POJO 类中布尔类型变量不要加 is 前缀，否则部分框架解析会引起序列化错误。**

说明：在本文 MySQL 规约中的建表约定第一条，表达是与否的值采用 is_xxx 的命名方式，所以，需要在<resultMap>设置从 is_xxx 到 xxx 的映射关系。

反例：定义为基本数据类型 Boolean isDeleted 的属性，它的方法也是 isDeleted()，RPC 框架在反向解析的时候，“误以为”对应的属性名称是 deleted，导致属性获取不到，进而抛出异常。

3 设计规范

3.1 平台设计器规范

1. 【强制】业务对象编码(对象标识)格式为：{开发商标识}_{模块}_{业务对象名}[_后缀]，长度不超过 36 位，整个系统中不允许重复，对象名为字符、_、数字。

   正例：isv_gl_voucher（其中 isv 为开发商标识）。

2. 【强制】业务对象中控件的标识（Key）在当前对象中不允许重复（注：此处标识包括对象唯一标识、单据体标识、字段标识、控件标识）。
3. 【强制】业务对象发布后不允许删除，业务对象的控件和字段原则上也不允许删除对于特殊情况需要删除重建的元数据应保证 FID 与原 FID 一致。
4. 【强制】业务对象表结构由数据库模型（PDM）设计定义（设计字段类型、长度、缺省值、为空属性和实体关系），二开表名以 tk_{开发商标识}为前缀，二开字段名以 fk_{开发商标识}为前缀并且必须与数据模型一致。
5. 【强制】业务对象不允许使用视图。
6. 【强制】设计器业务对象主键属性，必须和表定义主键一致。
7. 【推荐】字段标识（Key）由字符数字和下划线组成，长度 4-24。
8. 【推荐】ORM 属性命名真实反映元素具体含义，对应源代码实体属性名。
9. 【强制】ORM 实体命名为字符、_、数字，不允许有中文和特殊字符。
10. 【强制】设计器中表、字段、键、约束、缺省值的名称长度不超过 24 位（参照表设计规范）。

正例： isv_ecos_flexpanel（其中 isv 为开发商标识）。

3.2 扩展规范

当金蝶云苍穹提供的标准产品无法满足终端用户的个性化需求时采取的开发模式，通过对应用和单据的扩展，用户可以基于标准产品打造完全个性化的内容，这些扩展后的内容都是按租户严格隔离的，多个租户之间互不影响。扩展模式一般只提供给终端用户对标准产品进行个性化处理，当对单据进行扩展开发后，系统中原来访问原单据的地方会自动展示扩展后的单据，包括菜单、工作流，上下查等。

确保：支撑客户自定义业务需求、满足个性化功能、可持续化升级、保证二次开发兼容性、最小化升级包。

3.2.1 应用扩展

【强制】金蝶苍穹的所有类型的应用都支持扩展，但是只能垂直扩展，不能水平扩展，扩展过一次的应用不可再次扩展；扩展的应用可以继续向下扩展。

3.2.2 表单扩展

1. **【强制】源页面字段标识，是否必录，基础资料不允许修改，但可以修改字段名称；**

2. **【强制】源页面的字段标识不允许删除，如果不需要显示可以采用可见性处理。**

3.3 业务插件引用规范

【推荐】组件引用规范：尽量引用平台提供接口，不直接引用第三方 jar 包方法。

表单插件可引用 jar 包	平台-form-core 平台-entity-core 平台-form-mvc 平台-servicehelper
微服务可引用 jar 包	平台-entity-core 平台-servicehelper
操作插件可引用 jar 包	平台-entity-core

	平台-servicehelper
corelib 可引用 jar 包	平台-algo 平台-dataentity 平台-dbengine 平台-ormengine 平台-dtx 平台-dlock 平台-exception 平台-log 平台-mq 平台-serverscript mservice-schedule-api

【强制】组件引用规范：禁止直接添加并引用第三方 jar 包。

3.4 数据库规范

1. **【推荐】数据库对象命名规范**

对象类型	命名规则	最大长度	备注	示例
表	tk_[isv]_【系统标识_名称】	24	若为表体则再加上 entry 标识	TK_ISV_BAS_USER（其中 ISV 为开发商标识）
字段	FK_[isv]_名称	30	如果	FK_ISV_USERNAME （其中 ISV 为开发商标识）
主键	PK_[isv]_【系统标识_名称】	25		PK_ISV_BAS_USER
外键			禁止使用	

索引（包括唯一性索引	IDX_+[isv]_系统标识+_表\表缩写+字段\字段缩写	30	对于超长的表和字段可使用缩写，缩写建议为单词首字母组合	IDX_ISV_BAS_FNAME IDX_ISV_BAS_BCRE_FRULEID(单据代码规则分录表规则 ID 索引) IDX_ISV_BAS_BCRL_FRULEID（单据代码规则多语言表规则 ID 索引）

说明：如果是扩展的场景，则以上类型命名必须加上开发商标识加以区分，如果是自建的表单则可以不加。

2. **【推荐】字段类型规范**

数据类型	长度	Sql server	Oracle	MySQL	Postgresql
CHAR	[1,254]	CHAR[(n)]	CHAR[(n)]	CHAR[(n)]	CHAR[(n)]
VARCHAR	[1,4000]	VARCHAR[(n)]	VARCHAR2(n)	VARCHAR(n)	character varying(n), varchar(n)（最大 1G)
NCHAR	[1,2000]	NCHAR[(n)]	NCHAR[(n)]	VARCHAR(n)	CHAR[(n)]
NVARCHAR	[1,4000]	NVARCHAR[(n)]	NVARCHAR2(n)	VARCHAR(n)	VARCHAR TEXT
NCLOB	[1,1G]	NTEXT	NCLOB	TEXT	Text
SMALLINT	[-2^15,2~15-1]	SMALLINT	NUMBER(5)	SMALLINT	Smallint
INT INTEGER	[-2^31,2~31-1]	INT	NUMBER(10)	INT	Integer
DECIMAL	[$-10^{31+1,10}$31-1]	DECIMAL(23,10)	NUMBER(23,10)	DECIMAL(23,10) Decimal(19,6)	Decimal(23,10) Decimal(19,6)
DATETIME		DATETIME	DATE	DATETIME	DATE TIME TIMESTAMP

3.5 表结构设计规范

1. **【强制】**一个表的所有字段的总字节长度之和不能大于 8k（对于 LOB、Image 以及 nText 类型的长度不计算在内）。

2. **【强制】新建数据表时，必须定义主键（可以为非聚集的）。**

3. 【强制】新建数据库时，必须拥有聚集索引。

4. 【推荐】多语言字段单独放在一张表中。

3.6 字段设计规范

1. 【推荐】主从关系中，从表需要引用主表中某个字段，对应字段名称要求一致。

2. 【强制】一般数量、金额类型，必须使用精确数值类型;如：Decimal，禁止为空，指定默认值为 0；禁止使用 Double、Float、Money 等非精确类型。

3. 【强制】用 Decimal 数值类型时必须明确指定小数精度。

4. 【强制】nchar 字段的使用要非常慎重，存入的内容长度必须严格等于字段定义长度。

5. 【强制】类型的长度要根据实际情况确定，不能毫无根据的使用默认 255 位。

6. 【强制】not null 属性的字段，必须设置缺省值。

7. 【强制】数据类型，根据表中实际可能存储的值来确定，不能一律使用 int 类型。同样业务意义的字段定义必须一致。

4 开发规范

4.1 注释规范

1. **【推荐】注释的种类**

（1）以 /** 开始的行，可以产生 java doc。主要用于类、变量、方法的说明。

（2）以 /* 开始的行，不会产生 java doc，一般用于方法内部多行注释说明。

2. **【推荐】类的注释规范**

（1）类的版权

版权涉及到代码类的归属权，所以所有的类必须加上公司版权，方式：类名+版权。

/**

* @(#)<类名>

*

* XX 公司版权所有

*/

所属包和版权说明之间必须要有空行。

（2）类说明和申明

类说明和上面引用包之间必须要有空行，类说明和类申明之间不需要空行类说明必须包含描述@description、@author、@version、@createDate，如果有修改记录——包括修改人、修改时间、修改描述，加在作者和版本中间，同时上下都要有空注释行，同时如果存在多条修改记录时，修改记录与修改记录之间加入空行注释。类注释一般必须放在所有的

“import”语句之后，类定义之前，主要声明该类可以做什么，以及创建者、创建日期、版本和包名等一些信息。

正例：

```
/**
 * @projectName（项目名称）: project_name
 * @package（包）: package_name.file_name
 * @className（类名称）: type_name
 * @description（类描述）: 一句话描述该类的功能
 * @author（创建人）: user
 * @createDate（创建时间）: datetime
 * @updateUser（修改人）: user
 * @updateDate（修改时间）: datetime
 * @updateRemark（修改备注）: 说明本次修改内容
 * @version（版本）: v1.0
 */
```

正例：类注释需包含：创建人、创建时间、描述。

```
/**
 * @author: zhangsan
 * @createDate: 2018/10/28
 * @description: this is the student class.
 */
public class student{
```

.....

}

3. **【推荐】变量注释**

注释必须采用要生成 java doc 的注释方法。

正例：

```
/**
 * 变量描述
 */
Object obj;
```

4. **【推荐】方法注释**

方法的注释必须包括描述，根据实际的情况加入@param，@return、create date、@see。注释必须和前面的变量申明或者方法存在空行，方法的业务比较简单，方法内容程序小于 10 行，可以不加注释。注释和方法申明之间不需要空行。如果有修改记录----包括修改人、修改时间、修改描述，加在 data 和@see 之间，上下加入空行，同时如果有多天记录时，修改记录与修改记录之间加入空行。修改人、修改时间、修改描述的最后面必须一个<p>，如果有修改记录,修改记录的上一个描述最后面也必须加入一个<p>。

正例：

```
/**
 * @param num1: 加数 1
 * @param num2: 加数 2
```

```
 * @return: 两个加数的和
 */
public int add(int num1,int num2) {
    int value = num1 + num2;
    return value;
}
```

5. **【推荐】Java doc 常用标签**

（1）@author 作者名

（2）@version 版本号[,时间]

（3）@see 相关的连接类、方法

（4）@since 类或方法从什么版本开始存在的

（5）@param 对方法参数描述

（6）@return 对返回值的描述

（7）@throws 抛出异常

（8）@deprecated 不建议使用或将要废除的方法或类。使用时一定注意。

4.2 业务插件开发规范

1. **【强制】禁止在 beforeBinddata、afterBinddata 中修改数据对象。**
2. **【强制】业务代码禁止直接访问平台的元数据表 t_meta_xxx。**

3. **【强制】禁止继承标准产品表单上的插件。**

说明：该操作可能导致插件逻辑会被执行两次

4. **【强制】不允许禁用原厂插件。**
5. **【推荐】不使用废弃的接口方法。**
6. **【强制】禁止在插件 initialize()方法中注册控件事件和设置控件可见性等界面逻辑。**

说明：事件在 registerListener 中注册，界面控制逻辑在 afterBinddata 中处理

7. **【推荐】跨应用转换规则、反写规则及插件工程**：

（1）转换规则、转换插件应在源单所在应用；

（2）反写规则、反写插件应在目标单所在应用；

开发原则（推荐）：转换规则、转换插件、反写规则、反写插件，由源应用开发负责。

8. **【强制】引用对象创建或赋值必须保证对象类型一致。**

说明：通过平台创建的实体对象的引用对象是动态对象，对象的属性在不同实体上可能不一致，不同类型对象不能直接用赋值

正例：

```
public void createNewData(BizDataEventArgs e) {
   MainEntityType et = this.getModel().getDataEntityType();
   DynamicObject dataEntity = (DynamicObject) et.createInstance();
   BasedataProp prop = (BasedataProp) et.getProperty("customer");
```

```
        DynamicObject customerObj = (DynamicObject)
    prop.getComplexType().createInstance(); //正确
        prop.setValueFast(dataEntity, customerObj);
        e.setDataEntity(dataEntity);
      }
```

反例：

```
    public void createNewData(BizDataEventArgs e) {
        MainEntityType et = this.getModel().getDataEntityType();
        DynamicObject dataEntity = (DynamicObject) et.createInstance();
        BasedataProp prop = (BasedataProp) et.getProperty("customer");
        //customer 属性的实体类型与 bd_customer 实体类型不一样
      DynamicObject customerObj = (DynamicObject)
EntityMetadataCache.getDataEntityType("bd_customer").createInstance();
        ...
        prop.setValueFast(dataEntity, customerObj);
        e.setDataEntity(dataEntity);
      }
```

9. **【强制】实体元数据是单例对象，禁止在实体元数据缓存中获取到实体元数据后在内存中直接修改，应该先 Clone 再修改。**
10. **【推荐】查询一条数据是否存在时，改用 QueryServiceHelper.exist 方法，不用 queryOne 再判断。**

4.3 微服务开发及调用规范

1. **【推荐】二开微服务开发规范**

说明：

二开微服务必须由服务工厂注册定义才能使用，因此必须要有服务工厂类，服务工厂路由规则为：{isv 标识|公司标识}.{云 id}.{应用 id}.ServiceFactory，

正例：

新建微服务接口示例：hifi.ti.bo.mservice.xxxService （isv 标识为：hifi）

新建服务工厂示例：hifi.ti.bo.ServiceFactory（isv 标识为：hifi）

2. **【推荐】二开微服务调用规范**

static <T> T **invokeService**(String factoryQualifiedPrefix, String appId, String serviceName, String methodName, Object... paras)

（1）传入参数：

参数名称	参数	参数类型	是否必传
Factory 类限定前缀 如：服务工厂 isv.ti.bo.ServiceFactory 的 factoryQualifiedPrefix 为 isv.ti.bo	factoryQualifiedPrefix	String	是
应用 Id	appId	String	是
注册的服务名称	serviceName	String	是
调用服务方法	methodName	String	是
方法入参	paras	Object[]	是

（2）返回数据：微服务调用返回结果，Object

5 性能规范

1. **【强制】一般情况下，不建议插件开发使用多线程；如必须使用，可调用平台提供 ThreadPools 接口创建线程，禁止通过 jdk 自行创建。**

说明：为什么要使用平台封装后的线程池？

（1）统一管理和监控，比如监控总共创建了多少个线程，当前活动线程数。

（2）避免线程变量干扰，使用完线程返回线程池时，统一会线程变量清理和资源回收等工作。

（3）可传递 requestcontext 上下文到线程池线程中。

（4）相对 jdk 线程池做了更多优化，当线程池任务满时，不丢任务，更符合 erp 业务场景。

2. **【推荐】BusinessDataServiceHelper 和 QueryServiceHelper 使用规范**

服务	适用场景	缓存	数据结构	能否保存	备注
BusinessDataServiceHelper.loadSingle	取单张单据	无	单据结构一致	能	实时读取数据库，传入 pk 不存在时，会中断输出单条符合单据结构的数据包
BusinessDataServiceHelper.loadSingleFromCache	取单条基础资料	有	单据结构一致	否	优先从缓存中读取，缓存没有则实时取数据库，并压入缓存读取的数据包，经过缓存处理，少了状态信息
BusinessDataServiceHelper.load	取一批单据	无	单据结构一致	能	先根据条件取 pks，再根据 pks 实时取数，没有符合条件数据时返回空集合，不报错

BusinessDataServiceHelper.loadFromCache	取一批基础资料	有	单据结构一致	否	
QueryServiceHelper.query	取一批数据	无	plainObject	否	
QueryServiceHelper.queryOne	取一行数据	无	plainObject	否	
QueryServiceHelper.queryDataSet	取个别字段值	无	plainObject	否	流式取数，性能最快

说明：BOS 平台支持多数据库，所有业务数据必须通过 BusinessDataServiceHelper 和 QueryServiceHelper，使用时，根据不同的场景，比如能否使用缓存，是否需要保存等选择适合的服务方法进行查询。

3. **【强制】禁止在循环中调用 view.updateView()。**

   说明：可在循环结束后针对修改的数据进行局部刷新。

4. **【强制】禁止对大的数据包(>100)循环使用 model.setValue 更改数据。**

   说明：如需修改，建议使用 property.setValueFast。

   正例：

   for(DynamicProperty property : treeEntryProperties) {

   property.setValueFast(newObj,objProps.get(property.getName()).getValueFast(obj));}

   this.getView().updateView(entryKey); //针对修改数据局部刷新

5. **【强制】禁止在循环中访问数据库**

   说明：建议采用批量接口访问。

   正例：

   BusinessDataServiceHelper.loadFormCache(ids, "zsjt_originalbill");

   反例：

```
for (DynamicObject object : rows) {
    BusinessDataServiceHelper.loadSingle(object.get("id"), "zsjt_originalbill");
}
```

6. **【推荐】查询操作，尽量避免查询所有的字段**

   说明：

   1) 查询大数据量时应考虑增加更严格的过滤条件或分批查询处理；

   2) 应按需取数，除加载单据或基础资料外，取数必须指定字段。

7. **【强制】禁止在循环中访问 redis。**

   说明：如需大量存取缓存时，可以考虑减少查询次数或增加本地线程缓存减少对 Redis 压力。

8. **【推荐】频繁访问的数据应增加缓存，缓存的对象类型应考虑最大程度共享。**

   说明：可根据业务情况使用 loadFromCache 访问重复读取的数据，提高读性能，对同一类型数据应定义相同 DynamicObjectType(或相同的 selectProperties)，以达到最大程度数据共享。

6 数据访问规范

1. **【推荐】向 sql 传递参数时，避免使用拼装 sql 的形式，应使用参数的形式。**

2. **【强制】所有脚本必须使用 KSQL 语法，禁止使用方言。**

说明：特殊性能优化除外

3. **【强制】DataSet 使用完必须关闭。**

说明：DataSet 经过多次转换后会形成引用树，每个 DataSet 节点都会消耗或者引用资源，在使用结束后需要对叶子节点 Close。尽量少使用 copy，这个是有成本的，数据会保存到磁盘。

7 异常规范

1. 【推荐】统一使用 KDException，可自定义子异常。

2. 【强制】所有业务异常类必须使用 KDBizException。

3. 【推荐】异常处理方式：

   说明：

   （1）处理掉异常，一般多出现在 UI 层，使用统一的异常信息显示界面显示异常

   注意：

   a.界面显示的异常信息，应是业务语义，让用户知道下一步该怎么处理；

   b.不允许不作任何操作，直接隐藏掉异常的作法，有特殊理由，务必输出日志，并用注释说明原因；

   （2）将异常转换包装为另一种异常抛出。此时，应该将原始异常作为新异常的 cause 传入，以便异常信息的追踪；

   （3）将异常再次抛出。

4. 【推荐】异常实例化时，需要指明子异常信息，对于由其它异常引起的异常，需要将原异常作为 cause 传入，对于异常信息需要参数化的异常，还需要传入参数信息。

8 日志规范

说明：对于一些接口调用或者操作的关键信息，开发过程中应在适当的位置将日志输出，便于后续排查问题

1. **【强制】除一些特殊工具外，所有程序日志必须使用 kd.bos.logging.Log 框架，不允许使用第三方日志框架。**
2. **【强制】业务操作日志必须使用 BizLog.log 记录。**
3. **【强制】禁止在大循环中记录日志信息。**
4. **【强制】输出日志一定要先判断当前记录日志级别是否开启**

说明：该操作可以减少系统性能开销

正例：

```
// 引入类
import kd.bos.logging.Log;
import kd.bos.logging.LogFactory;
// 创建 logger
private final static Log logger = LogFactory.getLog(XXX.class);
// 使用 if (logger.isDebugEnabled()) {
   logger.debug("Debug info...");
}
```

5. **【推荐】日志输出，有异常时应带上异常对象。**

说明：如果没有异常对象信息，会导致对日志进行错误分析时无法定位错误原因

正例：必须使用 logger.error("错误分析描述", e) 形式；加上错误分析描述方便从错误

日志中查询特定问题的日志

反例：直接调用 Throwable.printStackTrace()

6. **【推荐】正确设置日志级别：Debug、Info、Warn、Error**

说明：如不是错误或者异常等，慎用 error 级别的日志

9 多语言规范

9.1 总体规范

1. 【强制】为了可以利用翻译平台支持程序提示语的翻译。切忌在一个工程中包含多个苍穹应用的提示语，不同苍穹应用的提示语应该放在不同的工程中。

2. 【推荐】图片的多语言：原则上图片和文字需要做分离，这样在实现多语言时只要把文字抽取出来翻译。如果文字嵌入图片中则需要整个图片重新设计多语言。

9.2 多语言字段

9.2.1 单据设计文本类型处理

1.【强制】多语言环境下，单据设计时注意事项：文本类型的字段，有需要显示不同国家语言文字的，则应**选择多语言文本控件**。

2.【推荐】多语言字段勾选“通用语言“属性，可解决的问题：即使在其他语言下没有维护该语言的译文，以该语言登录云苍穹后，此字段显示为中文而不是空：

9.2.2 多语言字段设计规范

【推荐】多语言表的字段长度设置：**字段长度根据业务实际情况设置，一般设置为 500。**另外，如果启用了通用语言的选项，对应的物理表中会增加一个文本字段，此字段的类型与长度与多语言表中的字段保持一致，此步骤由设计器自动完成。

录入多语言字段时，默认中文不能为空。否则翻译中心无法正常获取简体中文内容并进行翻译。

9.2.3 通用多语言自定义 SQL

【推荐】代码使用 orm 查询会自动处理通用语言字段，对于直接用 sql 查询多语言字段值，需要判断该字段是否开启了通用语言，然后若当前语言环境对应的值为 NULL 则取通用字段上的值。

```
//      mulilangfield为多语言字段，A为主表,B为多语言表，prop为多语言字段实体属性,通过case when then类型取值，或者全部取出，然后代码判断用哪个值；
        if(prop instanceof MuliLangTextProp && !prop.isDbIgnore){
         String sql = "select ..., case when B.mulilangfield is null "+
                                  "then A.mulilangfield" +
                                  "else B.mulilangfield" +
                                  "end" +
                      "from A left join B on B.FID=A.FID and B.flocaleid='zh_CN'";
        }else{
            //维持不变
        }
```

9.3 预插数据

9.3.1 预插数据规范

【强制】预插数据中不能填写分号，如果填写了分号，在制作多语言包时，分号会被识别为两个语句而截断，导致生成的语句在执行时报错

9.4 程序提示语

9.4.1 中文词条写法规范

1. **【强制】**程序提示语中的中文词条写法注意

（1）中文词条要纯正，不要汉语然后夹杂着英文符号

（2）词条的开头和结尾除非特殊原因最好不要留空格

2. **【强制】**中文词条中不要包含\n、\r 这样的转义字符，也不要包含 html 标签，比如，下面的中文词条：

"保存成功！\n 请检查这条记录！"

其中包含了\n 转义字符，在中文场景下加载正常，前端显示的时候会换行，然而，在其他语种下，会直接在界面上显示\n 而不是换行，这是因为多语言处理的时候经过了很多步骤，包括抽取、加载、工具转换、记录进入数据库、加载记录等过程，最后加载出来的转义字符会有多余的\。如果确实需要这样的换行效果，把上述的中文词条修改为如下的形式即可：

"保存成功！"+"\n"+"请检查这条记录！"

也就是把转义字符与其他中文拆开后再连接起来。

另外，html 包含了<>这样的特殊字符，在后续处理的过程中会报错，所以也需要把 html 标签与其它中文拆开后再连接起来，改造方法与转义字符的相同。

3.**【推荐】**代码中只包含需要翻译的中文内容。不要包含任何的格式、转移字符、特殊符号等。有可能导致译文显示错乱。

9.4.2 词条拼接处理规范

1. **【强制】**提示语必须是完整的句子，句子中的变量需要使用占位符替换。

通常遇到的使用场景如下：

场景 1：当代码中的提示语中只包含一个需要替换的变量时，可以使用%s 进行替换。

正例：

中文示例：“第 5 条行程。”

代码示例：String.format(ResManager.loadKDString("第%s 条行程。", "CslScheme_6", "fi-bcm-formplugin"), count)

说明：中文内容中的“5”是代码根据实际情况动态变化的，并且在这条语句中只存在一个变量需要动态替换。因此可以使用%s 进行替换，并通过 format 进行格式化。

场景 2：当在提示语中需要按照顺序位置替换多个参数内容时，可以使用%1$s，%2$s 按顺序使用。

正例：

中文示例：“第 2 行价格不能为空。”

代码示例：String.format(ResManager.loadKDString("第%1$s 行%2$s 不能为空。", "SrcPurListMustInput_1", "scm-src-common"), i + 1， displayName))

说明：要使用%1$s，%2$s，%3$s 等这种带参数位置的占位符，而不要使用%s，因为中文的语序跟其他语种的语序可能不同，这会导致翻译不准确或者翻译错误。在这个中文示例中，“2”和“价格”会随着录入人员实际录入的情况不同而发生变化。因此需要使用 2 个参数来进行变量替换。这时就需要使用%1$s 和%2$s 来进行替换。在代码示例中 i+1 用来替换行数，displayName 用来替换具体字段内容。

场景 3：

正例：

中文示例：“三月、四月、五月、六月未发现质检任务，请确认是否已开展质检工作“。

代码示例：

String xnamejoining = emptyList.stream().map(UnqualifiedCountResult::getXnames).collect(Collectors.joining("));

//将三月、四月、五月、六月先整合到一起。然后在作为参数使用%s 进行参数替换。

String.format(ResManager.loadKDString("%s 未发现质检任务，请确认是否已开展工作"，"SrcInput_1"，"scm-src-common")，month.toString()))

说明： 因为三月、四月等红色的第一部分内容是动态变化的，需要开发人员自己去分别获取。然后将获取到的三月、四月等内容拼接在一起。不能简单的使用场景 2 中的方式进行替换。

2. **【推荐】**使用 String.format（）进行带参数位置的格式化形式，可以满足格式化转换其他语言的需要。同时 ResManager.loadKDString 要放置到 format 的首个参数位置。

9.4.3 静态常量处理规范

1.【强制】程序提示语静态常量写法

kd.bos.permission.formplugin.AdministratorEditPlugin

以上 JAVA 类的中文静态常量：

```
private static final String MSG_CANCEL_EDITNEWADMIN = "新增管理员未保存，是否放弃保存？";
private static final String MSG_CURADMIN_SAVEORNOT  = "当前管理员未保存，是否放弃保存？";
```

用转换工具处理后变成：

```
private static final String MSG_CANCEL_EDITNEWADMIN = ResManager.loadKDString("新增管理员未保存，是否放弃保存？", "AdministratorEditPlugin_0", "bos-permission-formplugin");

private static final String MSG_CURADMIN_SAVEORNOT  = ResManager.loadKDString("当前管理员未保存，是否放弃保存？", "AdministratorEditPlugin_1", "bos-permission-formplugin");
```

这样的处理，在运行时不会根据不同的登录语言返回相应的译文。

```
this.getView().showConfirm(MSG_CANCEL_EDITNEWADMIN, MessageBoxOptions.OKCancel,
                           new ConfirmCallBackListener(eventInfo, this));
```

解决方案：增加一个获取提示语的方法：

```
private String getMsgCancleEditNewAdmin(){
    return "新增管理员未保存，是否放弃保存？";
}
```

原来使用静态常量的地方修改为调用获取提示语的方法

```
this.getView().showConfirm(getMsgCancleEditNewAdmin(), MessageBoxOptions.OKCancel,
                           new ConfirmCallBackListener(eventInfo, this));
```

程序提示语转换工具加工后的代码为

```
private String getMsgCancleEditNewAdmin(){ return ResManager.loadKDString("新增管理员未保存，是否放弃保存？", "AdministratorEditPlugin_0", "bos-permission-formplugin");  }
```

3. 【推荐】不能使用 static 来修饰中文静态常量，要按照推荐的方式对代码进行修改。开发完成后，可以使用硬编码检查工具来进行硬编码代码检查。保证代码不存在中文硬编码问题。

9.4.4 枚举类型处理规范

1. **【推荐】**枚举类也存在与静态变量类似的问题，建议改动办法如下：

比如有这样的一个枚举类（部分代码）：

```
public enum StateEnum {
    PASS("1","通过"),
    NOT_PASS("2","不通过");
…….
}
```

用转换工具处理后，变成（部分代码）：

```
public enum StateEnum {
    PASS("1",ResManager.loadKDString("通过", "StateEnum_0", "helloworld")),
    NOT_PASS("2",ResManager.loadKDString("不通过", "StateEnum_1", "helloworld"));
…….
}
```

但枚举类按照 Java 规范也只会初始化加载一次。如果切换语言，则不会根据当前登录或显示语言加载相应的译文。

改动办法：新增一个类（类名请命名为：MultiLangEnumBridge，否则硬编码检查工具会阻拦）：

```
public class MultiLangEnumBridge {
    private String description;
    private String resourceID;
    private String systemType;

    public MultiLangEnumBridge() {}

    public MultiLangEnumBridge(String description, String resourceID, String systemType) {
        this.description = description;
        this.resourceID = resourceID;
        this.systemType = systemType;
    }

    /**
     * 获取提示语
     *
```

```
     * @return
     */
    public String loadKDString() {
        return ResManager.loadKDString(this.getDescription(), this.getResourceID(),
this.getSystemType());
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getResourceID() {
        return resourceID;
    }

    public void setResourceID(String resourceID) {
        this.resourceID = resourceID;
    }

    public String getSystemType() {
        return systemType;
    }

    public void setSystemType(String systemType) {
        this.systemType = systemType;
    }
}
```

枚举类的修改：把枚举类中的中文字符，使用类 MultiLangEnumBridge 来进行初始化，并且调用它里面的方法获取当前译文，示例代码改造如下：

```
public enum StateEnum {

    PASS("1",new MultiLangEnumBridge("通过", "StateEnum_0", "helloworld")),
    NOT_PASS("2",new MultiLangEnumBridge("不通过", "StateEnum_1", "helloworld"));

    private String code;

    private MultiLangEnumBridge bridge = null;
```

```
    StateEnum(String code, MultiLangEnumBridge bridge){
        this.code = code;
        this.bridge = bridge;
    }

    public String getCode() {
        return code;
    }


    public static String getName(String code){
        if(code == null){
            return null;
        }
        for(StateEnum se:StateEnum.values()){
            if(code.equals(se.getCode())){
                return se.getName();
            }
        }

        return null;
    }

    public String getName(){
        return this.bridge.loadKDString();
    }

}
```

这样，根据枚举编码获取名称，可以按照当前登录语种返回对应的译文：

```
StateEnum.getName("1")
```

请按照以上规范改造包含中文的枚举类，如果采用其他修改办法可能会导致硬编码检查工具阻拦。

2.**【推荐】**Java 枚举类按照上面的规范进行修改就可以保证译文的正常显示。否则译文可能无法正常翻译和显示。

9.4.5 提示语编码规范

1.【强制】程序提示语为支持多语言，**必须不能使用硬编码，**而是要使用多语言资源文件。请注意，资源文件也就是**.properties 文件的编码一定要设置为 UTF-8**，否则可能会出现乱码。

如已有程序提示语硬编码，可通过硬编码检索替换工具进行处理

双击 ChineseHardCodeScanTool/ ChineseHardCodeScanUI.bat 打开硬编码检索替换工具。选择源代码根目录和日志输出目录。然后点击‘确定’，开始检索替换处理。

如果处理过程有异常，请根据日志修改源代码。处理成功后，会在源代码根目录产生一个目录 resources，在 resources 目录下会创建一个简体中文资源文件，工程编码_zh_CN.properties，编码格式为 UTF-8。同时把 Java 代码中的硬编码自动替换成如下的形式：

```
ResManager.loadKDString("自定义职能数量已超出限制：", "OrgBizFormPlugin_0",
"bos-org-formplugin")
```

代码中生成的内容与 properties 文件中的内容是一一对应的。

如果是使用 gradle 进行构建，则需要修改配置文件 build.gradle。

修改 copytolib 任务，增加一段

```
processResources{
    from('src/main/java') {
        include '**/*.properties'
        }
}
```

2.【推荐】推荐使用硬编码转换工具，对代码和 properties 进行修改。尽量不要手动修改带代码和 properties 文件。手动修改可能会导致参数不一致，或是因为编辑工具的编码不同，导致 properties 文件的编码格式不是标准的 UTF-8 格式。从而导致苍穹的翻译中心无法正确获取程序提示语，译文无法正常生成。

如果代码中的中文进行了修改，运行转换工具后。变更的中文也会自动更新到 properties 文件中。不需要开发人员手动替换。

9.4.6 硬编码校验方法

1.【推荐】程序提示硬编码改造完成后，可使用本地中文检查工具进行检查。以便确保代码符合多语言开发规范。

说明：中文检查工具可以向金蝶云苍穹国际产品申请。具体使用方法可以参照检查工具中的说明即可。

9.4.7 苍穹调度作业使用规范

1.**【强制】在使用苍穹调度任务时，如果涉及到程序提示语的使用。需要在任务中手动设置调度任务的语种。**

场景 1：在苍穹业务的开发中，业务开发人员可能会涉及到苍穹调度任务的开发和使用。并且也会使用到程序提示语。这时就需要特别注意，调用任务在执行时没有通过所在的上下文来获取语种，而是默认为中文语种，从而导致任务获取的程序提示语也是中文的。因此需要在初始化调度任务时，指定当前登录的语言。

```
JobInfo job = new JobInfo();
String OpName = ResManager.loadKDString("[illegible]:", "CalculateOutCostPlugin_8",
job.setName(OpName);
job.setJobType(JobType.REALTIME);
job.setParams(jobParams);
job.setAppId("cal");
job.setTaskClassname("kd.fi.calx.formplugin.calculate.out.CalculateOutTask");
job.setRunByUserId(Long.parseLong(RequestContext.get().getUserId()));
return JobClient.dispatch(job);
```

2.**【推荐】在调度任务中使用 setRunByLang(Lang.get())来设置运行语言。**

JobInfo job = new JobInfo();

```
Job. setRunByLang(Lang.get());
```

10 安全开发规范

10.1 敏感信息泄露

1. 【强制】代码中不能存在硬编码敏感信息，如账号、密码、http 外链、ftp 外链等；配置文件、SQL 脚本等文件中不能存在明文密码等敏感信息；

反例：

```
52
53    public YlGpsService() {
54        this.publicKey =
55        this.uid = '                    ;
56        this.pw =          ,
57    }
```

正例：

（1）将密码等敏感信息加密后存入 MC，代码再获取 MC 的值，通过解密得到具体的密码值，加密算法建议采取 AES256 或者 SHA256。

（2）所有的外链都必须写在配置文件，或者存放数据库中，产品默认的外链需要说明原因。

2. 【强制】页面不能明文显示敏感信息

反例：

正例：启用苍穹控件的内容显示为密码属性，并采用合适的加密算法，将敏

感信息加密后存库，需要使用时再进行解密使用。

10.2 第三方系统交互

以下规范基于苍穹的开发可以不关注，苍穹已经做了对应的防范并已通过安全测试，但与第三方系统交互，第三方 API 或开放 API 时，需要特别关注。

1. 【推荐】与第三方系统进行交互时，对传递的参数需要做校验，如果不合法，可直接拒绝请求

（1）如果是数字类型，判断是否为数字，如果不是，直接拒绝请求；

（2）如果是有格式的类型（比如日期、email、电话、身份证号码等等），判断是否符合格式，如果不符合格式，直接拒绝请求；

（3）如果是没有特殊格式的字符串类型，需要对字符串长度做校验。如果长度大于数据库该字段的定义长度，直接拒绝请求（比如姓名，数据库定义的是 varchar(12)，如果输入超过 12 个字符，直接拒绝请求）。

2. 【强制】禁止使用 Statement

说明：使用 PrepareStament，PrepareStament 可以有效防御 SQL 注入攻击，其原理是：MySQL 执行树会根据预设的参数做转义，把注入的 SQL 当作一个参数执行，而不会当作语句执行

3. 【强制】数据库 information_schema 权限不能开放给业务账号，各个业务账号只能访问自己业务的数据

说明：因为一旦某个漏洞被成功注入，information_schema 库将暴露所有库、所有表、字段的定义，给 SQL 注入者提供了便利。注入者不需要猜测表结构，就能轻松获取所有表的定义，进而轻松获取所有表的数据

10.3 XXE 外部实体注入

1.【强制】禁用 xml 外部实体；

2.【强制】过滤 xml 外部实体的关键字，过滤<!DOCTYPE>, <!ENTITY>, SYSTEM 等。

10.4 跨站脚本（XSS）

1. 【推荐】对数据将要置于的 HTML 上下文（包括主体、属性、JavaScript、CSS 或 URL）的所有不可信数据进行恰当的转义（escape），例如：对特殊符号< >进行 html 编码输出。

说明：基于苍穹控件的开发可以不关注以下防范措施，苍穹已经做了对应的防范并已通过安全测试，但使用自定义控件时，需要特别关注

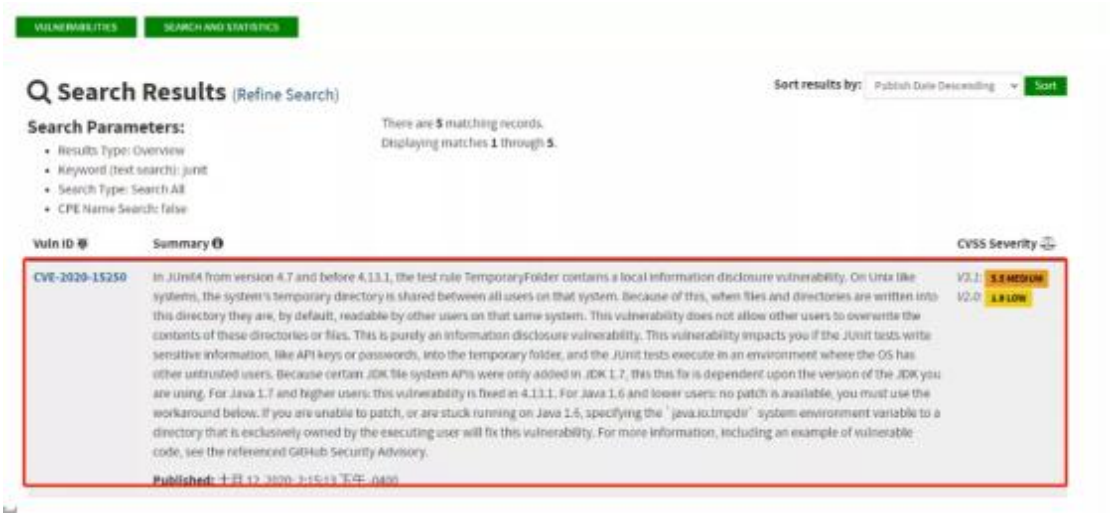
10.5 第三方组件漏洞

1. 【强制】使用第三方组件时，需要先检查组件的安全性

说明：检测第三方主件漏洞，可通过以下网址进行：

https://nvd.nist.gov/vuln/search

反例：存在漏洞的第三方组件：JUnit



正例： 不存在漏洞的第三方组件：lombok