广东省粮食信息管理平台及智能化粮库系统软件开发项目

系统设计开发指南 V1.0

项 目 名 称 ：广东省粮食信息管理平台及智能化粮库系统软件 开发项目

项目建设单位：广东省粮食和物资储备局

项目承建单位：紫光软件系统有限公司

项目监理单位：广州赛宝联睿信息科技有限公司

目 录

1 引言

1.1 目的

本文档是在粮食、物资管理等项目中使用的开发指南的基础上进行汇总整理，目的有以下两个：

- ✧ 统一项目中代码编写规范，以避免出现不一致的现象。
- ✧ 将原来设计的经验积累下来，实现设计经验的复用。

指导开发人员按统一风格编码提供，从而改进代码的可维护性、可读性，并从一定的程度上改进程序的性能。该指南也是代码评审的依据。

1.2 适用范围

1.1.1. 适用部门

适用于《广东省粮食信息管理平台及智能化粮库系统软件开发项目》软件开发部门。

1.1.2. 适用业务

《广东省粮食信息管理平台及智能化粮库系统软件开发项目》中 JAVA 程序语言的编码和代码评审活动。

1.3 说明

在本指南中，有部分建议条款，这些条款以*开始，并以斜体字表示，以便与强制条款区别。

1.4 术语和缩略语

术语/缩略语	解释
无	

2 编码基本要求

2.1 程序结构要求

2.1.1 程序结构清晰，简单易懂，单个方法的程序行数不得超过 150 行。

2.1.2 打算干什么，要简单，直截了当，代码精简，避免垃圾程序。

2.1.3 **尽量使用 Java API 和公共函数*。

2.1.4 **不要随意定义全局变量，尽量使用局部变量*。

2.2 可读性要求

2.2.1 可读性第一，效率第二(代码是给人读的)。

2.2.2 保持注释与代码完全一致。

2.2.3 每个源程序文件，都有文件头说明，说明规格见指南。

2.2.4 每个函数，都有函数头说明，说明规格见指南。

2.2.5 主要变量（结构、联合、类或对象）定义或引用时，注释能反映其含义。

2.2.6 处理过程的每个阶段都有相关注释说明。

2.2.7 在典型算法前都有注释，同时算法在满足要求的情况下尽可能简单。

2.2.8 利用缩进来显示程序的逻辑结构，缩进量一致并以 Tab 键为单位，定义 Tab 为 4 个字节。

2.2.9 循环、分支层次不要超过 3 层。

2.2.10 注释可以与语句在同一行，也可以在上行。

2.2.11 空行和空白字符也是一种特殊注释。

2.2.12 一目了然的语句不加注释。

2.2.13 注释的作用范围可以为：定义、引用、条件分支以及一段代码。

2.2.14 注释行数（不包括程序头和函数头说明部份）应占总行数的 1/5 到 1/3

2.2.15 常量定义（DEFINE）有相应说明。

2.3 结构化要求

2.3.1 禁止出现两条等价的支路。

2.3.2 禁止 goto 语句。

2.3.3 用 if 语句来强调只执行两组语句中的一组。禁止 else goto 和 else return。

2.3.4 用 case 实现多路分支。

2.3.5 避免从循环引出多个出口。

2.3.6 不使用条件赋值语句。

2.3.7 不要轻易用条件分支去替换逻辑表达式。

2.4 正确性与容错性要求

2.4.1 程序首先是正确，其次是优美

2.4.2 无法证明你的程序没有错误，因此在编写完一段程序后，应先回头检查。

2.4.3 改一个错误时可能产生新的错误，因此在修改前首先考虑对其它程序的影响。

2.4.4 所有变量在调用前必须被初始化。

2.4.5 对所有的用户输入，必须进行合法性检查。

2.4.6 不要比较浮点数的相等，

如： 10.0 * 0.1 == 1.0 ， 不可靠，用差值比较。

2.4.7 程序与环境或状态发生关系时，必须主动去处理发生的意外事件，如文件能否逻辑锁定、打印机是否联机等,对于明确的错误,要有明确的容错代码提示用户。

2.4.8 单元测试也是编程的一部份，提交联调测试的程序必须通过单元测试。限对外接口。

2.4.9 尽量使用规范的容错语句.

例：

```
try {
    ……
} catch (Exception e) {
    // TODO: handle exception
} finally {
    ……
}
```

2.5 可重用性要求

2.5.1 重复使用的完成相对独立功能的算法或代码应抽象为服务或类。

2.5.2 服务或类应考虑 OO 思想，减少外界联系，考虑独立性或封装性。

3 目录规划(Directory Layout)

每个命名空间产生一个目录(如 MyProject.TestSuite.TestTier 用 MyProject/TestSuite/TestTier 作为路径)，这样使得命名空间映射到目录规划比较容易(对新项目采用此方法，原有项目修改时可以不改)。

4 命名规范

4.1 基本原则

1. 使用一看就明白含义的词汇命名，尽量不使用缩写，尽量采用英文单词表达用途，难以表达的用拼音替代，如果采用拼音则用全拼或马上添加相应的说明注释，说明对应的中文。
2. 在名称中建议不要包含下划线“_”。

4.2 类

1. 采用 Pascal 大小写的命名方式，每个单词的首字母大写。
2. 名字应该能够标识事物的特性，而且尽量不使用缩写，除非是众所周知的。
3. 名字的校验正则表达式为：^[A-Z][a-zA-Z0-9]*。

例：

```
public class FileStream
public class DBOperate
public class CustomInfo
```

4.3 接口

和类命名规范相同，唯一区别是接口在名字前加上“I”前缀。

4.4 抽象类

和类命名规范相同，区别是要在类的前面加上前缀 Abstract 或者以 Factory 结尾。

抽象类的校验正则表达式为：^Abstract.*$|^.*Factory$

例：

```
abstract  public class AbstractBuildBill
```

4.5 变量

1. 采用 Camel 命名方法，首个单词的首字母小写，其余单词的首字母大写。
2. 即使对于可能仅出现在几个代码行中的生存期很短的变量，仍然使用有意义的名称。仅对于短循环(仅限一层循环)索引可以使用单字母变量名，如 i 或 j。

例如：

```
int userID;
string userName;
```

3. 变量的校验表达式为：^[a-z][a-zA-Z0-9]*$
4. static final 类型 的变量应全部大写，参照常量的命名规范

   校验表达式为：^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*$

4.6 方法

方法名 Camel 命名方法，首个单词的首字母小写，其余单词的首字母大写。

例如：

```
public boolean saveData()
```

方法名称的校验表达式为：^[a-z][a-zA-Z0-9]*$

4.7 参数

采用 Camel 命名方法，首个单词的首字母小写，其余单词的首字母大写。

4.8 常量

常量由字母数字和下画线组成，所有字母全部大写。

常量的校验表达式为：^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*$

4.9 属性

使用 Camel 命名方法，首个单词的首字母小写，其余单词的首字母大写。属性遵循 JavaBean 规范，自动产生 get/set 方法对。

示例：

```
/**
  * 操作名称.
  */
 private String name;
/**
  * 获取name.
  *
```

```
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * 设置 name.
     *
     * @param newname
     *          the name to set
     */
    public void setName(final String newname) {
        this.name = newname;
    }
```

4.10 包

包名全部小写，单个包名不宜过长，长了请使用缩写。推荐使用公司名称创建顶级的包名，再嵌套产品的名称作为二级包名，后面再按产品中的领域进行命名。

例如：ygsoft.platform.dj.bc

ygsoft.platform.dj.bo

5 注释规范

在大多数情况先，均使用 Html 格式进行，其规格符合 Microsoft 的标准。但是单行注释（用于注释单个语句块）、多行注释（用于注释多个语句块）、outline 注释（也就是使用////进行的注释，用于暂时性的注释）可不必要使用 Html 格式。

5.1 文件头

```
/*
 * Copyright 2018-2025 ShenZhen SYDATA Ltd. All Rights Reserved.
 */
```

5.2 类

类的注释格式如下：

```
/**
* 业务对象基类实现.<br>
*
* @author bw <br>
* @version 1.0.0 2009-6-22<br>
* @since JDK 1.4.2.6
*/
public class BusinessObject implements IBusinessObject {
}
```

5.3 方法

在类的方法声明前必须以以下格式编写注释：

```
    /**
     * 获取统一选择控件的业务数据.
     * @param dataContext 上下文信息.
     * @param params 统一选择控件参数.
     * @return 统一选择控件的业务数据.
     */
    Public SelectorResult execute(final DataContext dataContext, final
IBusinessDataSetObject params) {
}
```

5.4 类属性、成员变量

在类的属性必须以以下格式编写属性注释：

```
    /**
     * DAO变量.
     */
    private IYHSelectDAO yhSelectDAO;
```

5.5 代码间注释

1. 代码间注释分为单行注释和多行注释：

    单行注释：

    //<单行注释>

    多行注释：

    /*多行注释 1

    多行注释 2

```
多行注释 3*/
```

2. 注释单独成行，避免直接写在代码后面，便于代码合并与阅读。

   例：

   错误的注释位置

```
uniSelectService.setUniSelectorBC(uniSelectorBC);  //  设置统一选择 BC Bean
```

   正确的注释位置

```
// 设置统一选择 BC Bean
uniSelectService.setUniSelectorBC(uniSelectorBC);
```

3. 代码中遇到语句块时最好添加注释（if,for ,……）,添加的注释必须能够说明此语句块的作用和实现手段（所用算法等等）。注释要写在语句块的最上面，不能破坏代码块的完整性。

   例：

   错误的注释位置。

```
if(…) {
    //注释
    ……
} else {
    //注释
    ……
}
```

   正确的注释位置：

```
//注释
if(…) {
    ……
} else {
    //注释
    ……
}
```

4. 在编写代码期间要养成写注释的习惯。一般对于单条语句注释写在这条语句代码的上面。

如：

```
    // 此处设置构造树属性和Parent属性.
    treeResult.setTreeAttribute(DXLXSelectDAOConst.FIELD_ID);

treeResult.setParentTreeAttribute(DXLXSelectDAOConst.FIELD_PID);
    // 此处设置树节点的显示内容和值属性.
    treeResult.setNameAttribute(new
String[]{DXLXSelectDAOConst.FIELD_MC});
    treeResult.setValueAttribute(new
 String[]{DXLXSelectDAOConst.FIELD_ID});
```

5. 在已入库的代码基础上增加代码时，必须增加注释，注释必须包含代码新增人、增加时间、单据号（如果是根据单据新增的）以及相关说明信息，格式如下：

```
/**
 * APPEND R3799  @author wanghuojun <br>
 * add new code 2009-12-24 上午08:47:35 <br>
 * 在此添加新增代码的说明  <br>
 */
//TODO 在此添加新代码   <br>
//append new code end <br>
```

6. 删除已入库的代码时，必须增加注释，注释必须包含代码删除人、删除时间、单据号（如果是根据单据删除的）以及相关说明信息，格式如下：

```
/**
 * DELE F13899 @author wanghuojun <br>
 * delete code 2009-12-24 上午08:56:09 <br>
 * 在此添加删除代码的注释 <br>
 */
```

7. 修改已入配置库的代码时，必须在改动处增加注释，注释必须包含修改人、修改时间、单据号（如果是根据单据修改的）以及相关说明信息，格式如下：

```
/**
 * MODIFY W3899 @author wanghuojun <br>
 * modify code 2009-12-24 上午08:57:20 <br>
 * 在此添加修改代码的注释 <br>
 */
```

6 可读性/排版

6.1 空格/空行的使用

1. 使用空白行。空白行将语句划分成有意义的段落，以增加可读性。

1.1 为了便于阅读，二元运算符（如“+”、“-”、“*”、“/”、“>”、“==”、“<”、“>=”、“<=”等）以及赋值符（“=”）的前后必须有空格存在。

1.2 当定义构建器、定义方法、调用方法、调用父类构建器时，参数之间必须以空格隔开，格式如下：

```
final IYGSQLQuery query = getYGSQLQuery(dataContext, querySql,
params);
```

1.3 左大括号（“{”）前面必须有空格，右大括号（“}”）如果后面有其他代码也应以空格隔开。

如：

```
    private void setParameters(final SelectSource sourceDes,
            final SelectorDescription selectorDescription) {
        ……
    }

    try {
        result.setListItemClass(Class.forName(dataClassName));
    } catch (Exception e) {
        throw new GRISRuntimeException(e);
    }
```

1.4 每个 JAVA 类文件必须以空白行结束，因为版本控制器（如 CVS）遇到结尾没有新空白行的文件会出现警告。

1.5 *所有的代码元素（比如方法，类属性，构造器等等）之间最好使用空白行分隔，如：。

```
/**
 * DAO变量.
 */
private IUniSelectorDAO uniSelectorDAO;

/**
 * 数据来源业务处理对象.
 */
private IUniSelectorDataSource uniSelectorDataSource;
```

2. 空白行规则-不使用空白行，不必要的空白是禁止使用的。

2.1 当定义构建器、定义方法、调用方法、调用父类构建器时与接下来的左括号之间不需要空格。

如：

```
StringUtil.isNotEmptyString(dataClassName)
```

2.2 在一元运算符（“~”、“--”、“++”、“!”）、正负号（“+”、“-”）以及“.”等这些符号的后面不允许出现空白区域。

2.3 在“;”、“--”、“++”、“.”这些符号的前面不允许出现空白区域。

2.4 左括号和后一个字符之间不应该出现空格，同样，右括号和前一个字符之间也不应该出现空格。

如：

```
callProc( parameter ); // 错误
```

```
callProc(parameter); // 正确
```

2.5 使用空白行分割语句时仅仅需要一行就可以了，不需要多行空白

2.6 单行注释后面需要跟代码，而不能是空白行。

6.2 缩进

1. 缩进应该是每行 4 个空格. 在使用不同的源代码编辑工具时，开发人员不需要对 Tab 字符的长度进行设置，避免 Tab 将来扩展为不同的宽度。
2. 不要在源文件中保存 Tab 字符('\\t')。

6.3 数组定义

JAVA 中的数组定义支持两种格式(Java-style 和 C-style),为了方便阅读，统一采用 Java-style 形式。

如：

采用下面 Java-style 形式

```
public static void main(String[] args)
```

而不采用下面 C-style 形式

```
public static void main(String args[])
```

6.4 长整形定义

长整形应该用“L”而不用“l”来标示，因为小写字母“l”有点类似阿拉伯数字 1。

如：

```
final long longNum = 1234L;
```

6.5 代码块（Blocks）

1. 编写复合代码块语句（如 if/else、try/catch 等）时，后部分的代码（下例中的 else 部分）应紧接前一代码块，且位于同一行内。

如：

```
if (result != null) {
    allSelectorDescs = (IBusinessDataSetObject) result;
```

```
} else {
    allSelectorDescs = loadUniselectorDescToCache();
}
```

2. 编写复合代码块语句（如 if/else、for、do/while、switch 等）时，即使主体只有一行代码，也应包含在大括号{}之中。

   如：

   上例代码如果写成这样则是不允许的

```
if (result != null)
    allSelectorDescs = (IBusinessDataSetObject) result;
else
    allSelectorDescs = loadUniselectorDescToCache();
```

6.6 布尔语句

1. 为了增强代码可读性，尽量避免使用 inline Conditionals，及一行代码中出现条件判断并返回值。

   如：

```
// 下面一行代码中存在条件判断并返回值，是不允许的
final String b = (a == null || a.length() < 1) ? null : a.substring(1);
```

2. 如果方法的返回值是 Boolean 类型时，尽量做到简化。

   如：

```
if (valid()) {
    return false;
} else {
    return true;
}
```

   上面的代码可直接简化成

```
return !valid();
```

7 可维护性

7.1 代码复杂度

1. 单个 JAVA 类文件的最大长度不能超过 2000 行（含注释部分）。
2. 单行长度不能超过 120 个字符, tab 宽度为 4（不允许修改）。

3. 每个方法的最大长度不超过 150 行。
4. 方法或者构造器的参数个数不能超过 7 个，如果超过，则建议将多个参数封装成一个对象。
5. 匿名类的代码行数最大不能超过 20。
6. 一个条件表达式中最多只能包含三个条件，多了请使用一个方法替换。
7. 数据抽象耦合度是描述对象之间的耦合度的一种代码度量。DAC 度量值表示一个类中有实例化的其它类的个数,最大推荐为 7。
8. 一个类被其他类引用的次数推荐最大为 20（工具类除外）。
9. 一个方法的圈复杂度推荐最大为 10，多了请重构。圈复杂度可以通过决策点的数量来计算，具体计算方法是：从 1 开始，一直往下通过程序，一旦遇到 if、while、for、||、&&时就加 1，如果有 switch，每个 case 加 1。
10. NPath 度量值表示一个方法内可能的执行路径的条数,最大值推荐不超过 200。
11. 没有注释的代码行数,一个方法最多为 50 行，一个类最多为 1500 行，一个文件最多为 2000 行。
12. 方法的分支语句嵌套深度必须小于等于 3。
13. 方法的循环语句嵌套深度小于等于 3。

7.2 代码重复度

编写代码时要严格限制代码的重复程度，避免类之间代码内容大部分相同，除了 import 语句忽略之外，其他所有语句（包括 JavaDoc 内容），方法间的空白行等都属于比较范围，且最大连续相同行数不允许超过 12。相同代码可考虑采用封装公共方法进行调用。

7.3 Import 引入规范

1. 尽量避免通过*符号引入包下面所有的类，因为引入所有的类意味着包与包之间关联过于紧密。
2. 不能导入 sun.*包，因为程序默认已经引入了。
3. 不能引入多余的包，如多次引入相同的包、引入 java.lang 下面的类、引入同包下面的类。
4. 不能引入没有使用过的包。

7.4 类设计规范

1. 避免构建器为私有的类定义成 final 类型，这样的话无法正确使用。
2. 确保Utility类(仅含static方法或static代码域的类)的构建器为private或者 protected 类型。
3. 接口是一种类型，不能没有定义一个方法而全部是常量。
4. 只有 static final 类成员可以用 public 修饰，除非类成员设定为 protectedAllowed 或者 packageAllowed。
5. 确保普通类的构造器和方法的参数是 final 类型，接口不受此指南约束，因为接口没有方法体，不会更改参数。

7.5 特殊方法重写

1. 当你重写 equals()方法时必须重写 hashCode()方法。
2. 当重写 clone()方法时，确保事先调用 super.clone()方法。
3. 当重写 finalize()方法时，确保事先调用 super. finalize()方法。

7.6 修饰符顺序

修饰符的顺序如下：

1. **public**
2. **protected**
3. **private**
4. **abstract**
5. **static**
6. **final**
7. **transient**
8. **volatile**
9. **synchronized**
10. **native**
11. **strictfp**

7.7 避免空循环和空语句

1. 避免在代码中出现空循环语句，如类似下面的代码是不允许的。

```
for (int arrayNum = 0; arrayNum < array.length; arrayNum++) {
}
```

2. 避免在代码中出现空语句，空语句对程序执行的输出没有任何作用，留在代码中还会给后面的维护人员造成很大的困扰，如类似下面的代码是不允许出现的。

```
Math.abs(num);
```

7.8 Switch 语句容错

选择语句 switch 最后需要提供 default 选项对选项之外的情况进行容错处理。

7.9 数字和字符串

禁止在逻辑代码中直接使用数字和字符串(不包含业务含义且一看就明白的简单整数除外)。在代码中直接使用数字和字符串，降低了代码的可维护性。后期维护者很难了解数字和字符串的含义，也导致了需要在多处修改程序。使用数字和字符串时，应首先在常量类中定义一个常量，再使用这个常量。

7.10 表达式的运算优先级

算术表达式和条件表达式都必须显式说明优先级，而不采用缺省优先级。

示例_算术表达式 ：

```
int x = a + y * b / c % z - d;
```

对于中间优先级相同的乘法，除法，模运算不能仍然采用缺省优先级可以唯一确定一个计算顺序，但是指南要求使用括号显式表达处理，增强可读性。比如上述表达式可以是

```
int x = a + (((y * b) / c) % z) - d;      // 和默认的优先级一致
int x = a + (y * ((b / c) % z)) - d;      // 和默认的优先级不一致
```

示例_条件表达式：

```
if (x || y && z && a || b) {
  ……
}
```

因为按照&&的优先级大于||，运算的顺序是：

x

```
y && z
y && z && a
x|| y && z && a
x || y && z && a ||b
```

可读性不好，必须使用括号显示表示其优先级

```
if (x || (y && z && a) || b) {
   ……
}
```

7.11 清除未引用的局部变量

为了使编写的程序结构清晰，便于阅读和维护，程序中没有引用到的变量要坚决清除。

7.12 不要在方法体内修改参数的引用

不要在方法体内修改参数的引用。对于参数为对象的，如果程序逻辑需要，可以在方法体内修改参数的属性值，但要在方法的注释中予以说明。

7.13 类实例化选择

当类提供了工厂类实例化时，最好调用工厂类实例化，而非调用构建器实例化，建议提供了工厂构造函数的类定义私有或者保护的构造函数。

8 性能

8.1 循环性能

在循环操作，特别是遍历次数比较多的循环中，尽量不要在循环体内构造和释放对象，可把构造和释放对象的操作放在循环体外。

8.2 String 和 StringBuffer

在处理 String 的时候要尽量使用 StringBuffer 类，StringBuffer 类是构成 String 类的基础。String 类将 StringBuffer 类封装了起来，（以花费更多时间为代价）为开发人员提供了一个安全的接口。

在构造字符串的时候，应该用 StringBuffer 来实现大部分的工作，当工作完成后将 StringBuffer 对象再转换为需要的 String 对象。比如：如果有一个字符串必须不断地在其后添加许多字符来完成构造，那么应该使 StringBuffer 对象和它的 append()方法。如果用 String 对象代替 StringBuffer 对象的话，会花费许多不必要的创建和释放对象的 CPU 时间。

另外，如果不在多线程中使用 StringBuffer 时，建议使用 YGFastStringBuffer。

8.3 尽量避免使用 synchronized 关键字，如果一定要是使用，需要得到项目组内部的认同

避免使用关键字 synchronized，这将有助于避免死锁。

8.4 final 应用场景

根据程序上下文环境，Java 关键字 final 有“这是无法改变的”或者“终态”的含义，它可以修饰非抽象类、非抽象类成员，以及变量，合理利用 final 修饰符，能大大提高程序的运行效率。

1. final 类，final 类不能被继承，final 类的成员方法默认都是 final 的，在设计类时，如果这个类不需要有子类，类的实现细节不允许改变，并且确信其不会被扩展，那么就设计成 final 类。

2. final 方法，final 方法不允许子类覆盖重写，并且其运行效率比普通类高，因为编译器在遇到有调用 final 方法的情况时，会转入内嵌机制。

3. final 修饰的变量有三种：静态变量（与 static 一起修饰）、实例变量和局部变量，这些变量都不允许调用时对其值做任何修改。

4. final 参数，由于我们在指南中不允许修改方法或构造器的参数引用（详见 7.12），所以方法和构造器的入参均应该为 final 类型。

8.5 异常对性能的影响。

异常对性能影响很大，程序中不要以异常来控制程序的执行流程，尽量不要抛出新异常（捕获的异常除外）。

8.6 合理安排分支结构中分支的顺序。

1． and、or 的运算顺序：

如果判断语句中含有 and(&&)或 or(||)运算，程序执行的过程中并不是条件都去判断的。假如在一个判断语句中有 N 个条件，在执行 and 运算中，当执行到第 n 个为否时，就不再往下判断。同理，在执行 or 运算时，执行到第 n 个为真时，就不再执行。也就是说，在上述两种情况下，他们只执行前 n 个函数（如果，您把函数的返回值作为判断条件）的话，而后面 N-n 个不一定执行。合理安排判断语句中条件的顺序可提高程序的运行效率。

and 运算应该将为结果为 false 概率高的条件放在前面；

or 运算应该将为结果为 true 概率高的条件放在前面；

2． 根据运算复杂度安排条件顺序：

将运算复杂度低的条件安排到判断语句的前面，就会降低运算复杂度高的条件的命中概率，从而提高程序的运算效率。

3． **合理安排 if… else if…else 分支的顺序*。

将命中率高的分支放到前面。如果各分支的判断语句运算复杂度相差较大，建议将运算复杂度小的分支放到前面。

8.7 用变量存放方法的返回值。

如果需要多次使用某个方法的返回值(每次返回值不变)，这时要用一个变量存放返回值。在使用返回值之前调用这个函数，并将返回值存放在变量中，需要使用这个函数返回值的地方不再调用函数，直接使用存放返回值的变量。

8.8 返回结果应按需返回

在接口或服务返回结果的时候，需要考虑是否在同一个线程，如果不是在一个线程内，如服务端请求和微服务请求的场景，需要重点考虑返回值的大小范围，

不需要的属性不返回，数据量不能超过一页的数据范围。

8.9 导入导出相关范围控制

导入时需要控制文件的大小和记录数的大小，不要没有范围控制。导出时也需要考虑数据量的大小和导出文件是否能够容纳这么多数据。避免由于文件太大和数据大多引起的系统问题。

8.10 事务大小范围控制标准

避免把在类似循环操作或调度轮询包装成为一个大事务，应该尽量保证事务的独立性和轻便性。

8.11 日志输出对象大小控制

日志输出除非需要按照指南等级输出外，还要严格遵循输出的时候不能把大的对象数据，如数据集、列表等。类似这类的输出，应该只输出关键行和需要关注的值。

9 规范检查

本指南中的代码编写规则，采用 eclipse 的 CheckStyle 作为代码分析工具（导入了代码检查规范）。对于不易自动化检查规则部分，通过代码评审、代码走读等方式进行检查。