**SmartWork开发规范**

**文档历史**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **日期** | **作者** | **描述** |
| **V1.0** | 2015-12-14 | 王炳翰、罗亚 | 初版 |
| **V2.0** | 2019-02-19 | 田恬 | 优化标题名，增加部分规范 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1 排版规范 5](#_Toc1492729)

[1.1 Eclipse格式化代码设置 5](#_Toc1492730)

[1.2 空行 5](#_Toc1492731)

[1.3 换行 6](#_Toc1492732)

[1.4 空格 7](#_Toc1492738)

[2 注释规范 9](#_Toc1492753)

[2.1 注释量 9](#_Toc1492754)

[2.2 注释规则 9](#_Toc1492755)

[2.2.1 文件注释 9](#_Toc1492756)

[2.2.2 函数注释 10](#_Toc1492757)

[2.2.3 代码注释 10](#_Toc1492758)

[2.2.4 必写注释 12](#_Toc1492759)

[3 命名规范 12](#_Toc1492760)

[3.1 包的命名 12](#_Toc1492761)

[3.2 类的命名 13](#_Toc1492762)

[3.3 方法的命名 13](#_Toc1492763)

[3.4 变量的命名 13](#_Toc1492764)

[3.5 常量的命名 14](#_Toc1492765)

[3.6 参数的命名 14](#_Toc1492766)

[3.7 数组变量的命名 14](#_Toc1492767)

[3.8 命名要求 14](#_Toc1492768)

[3.9 命名禁止 15](#_Toc1492769)

[4 编码规范 16](#_Toc1492770)

[4.1 可读性 16](#_Toc1492771)

[4.1.1 优先级 16](#_Toc1492772)

[4.1.2 枚举或者常量 16](#_Toc1492773)

[4.1.3 相邻原则 17](#_Toc1492774)

[4.1.4 避免炫技 17](#_Toc1492775)

[4.2 JAVA日志打印输出 17](#_Toc1492776)

[4.2.1 LogUtilities的使用 18](#_Toc1492777)

[4.2.2 Log日志管理 18](#_Toc1492778)

[4.2.3 DS表日志记录 18](#_Toc1492779)

[4.2.4 对象日志记录 19](#_Toc1492780)

[4.2.5 操作日志 20](#_Toc1492781)

[4.2.6 日志查看 21](#_Toc1492782)

[4.3 国际化 21](#_Toc1492783)

[4.3.1 尽可能的使用国际化 21](#_Toc1492784)

[4.3.2 国际化类命名规范 21](#_Toc1492785)

[4.3.3 国际化类常量及MessageID命名 22](#_Toc1492786)

[4.3.4 国际化包名应和国际化类名一致 22](#_Toc1492787)

[4.3.5 国际化方法 22](#_Toc1492788)

[4.4 测试代码 24](#_Toc1492789)

[4.5 性能优化 24](#_Toc1492790)

[4.6 类命名唯一 24](#_Toc1492791)

[4.7 方法设计 25](#_Toc1492792)

[4.8 类设计 25](#_Toc1492793)

[4.9 私有化 25](#_Toc1492794)

[4.10 Import 语句 25](#_Toc1492795)

[4.11 “大对象” 26](#_Toc1492796)

[4.12 内部类 26](#_Toc1492797)

[4.13 注释强调 26](#_Toc1492798)

[4.14 对象清除(资源泄露) 26](#_Toc1492799)

[4.15 对象集合 27](#_Toc1492800)

[4.16 Interfaces 27](#_Toc1492801)

[4.17 构造器 27](#_Toc1492802)

[4.18 尽量新建类，避免继承 27](#_Toc1492803)

[4.19 避免使用“魔术数字” 27](#_Toc1492804)

[4.20 警惕“分析瘫痪” 28](#_Toc1492805)

[4.21 警惕“过早优化” 28](#_Toc1492806)

[4.22 谨惕“巨大对象综合症” 28](#_Toc1492807)

[5 常见的编码错误 28](#_Toc1492808)

[5.1 字符串判断 28](#_Toc1492809)

[5.2 数组越界 28](#_Toc1492810)

[5.3 慎用全局变量 29](#_Toc1492811)

[5.4 数据库查询 29](#_Toc1492812)

[6 Layout布局 29](#_Toc1492813)

[6.1 AbsoluteLayout 29](#_Toc1492814)

[6.2 FillLayout 29](#_Toc1492815)

[6.3 GridLayout 30](#_Toc1492816)

[6.4 FormLayout 31](#_Toc1492817)

[7 容器使用 31](#_Toc1492818)

[7.1 面板 31](#_Toc1492819)

[7.2 分组框 31](#_Toc1492820)

[7.3 选项卡 32](#_Toc1492821)

[8 控件使用 32](#_Toc1492822)

[8.1 控件使用原则 32](#_Toc1492823)

[8.2 控件边框 32](#_Toc1492824)

[8.3 控件布局 32](#_Toc1492825)

[8.4 Label标签 33](#_Toc1492826)

[8.5 Text文本框 34](#_Toc1492827)

[8.6 Combo下拉框 35](#_Toc1492828)

[8.7 SmartEdit多功能控件 35](#_Toc1492829)

[8.8 ToolBar和Button 35](#_Toc1492830)

[8.9 基础面板 37](#_Toc1492831)

[9 PD规范 37](#_Toc1492832)

[9.1 PROCESS DESIGNER建表规范 37](#_Toc1492833)

[9.1.1 表名规范 37](#_Toc1492834)

[9.1.2 字段命名规范 37](#_Toc1492835)

[9.1.3 字段类型及长度选择 38](#_Toc1492836)

[9.1.4 常用对象 38](#_Toc1492837)

[9.1.5 培训PD建表规范 39](#_Toc1492838)

**第一部分 编码规范**

# 排版规范

## Eclipse格式化代码设置

window-preferences–>java-code Style-formatter

编辑，在tab的Line Wrapping设置 最大的line width为120即可。

下载指定的Code Templates和Formatter导入到Eclipse里即可。

可以使用鼠标选中代码块之后，然后使用Ctrl+Shift+F 快捷键 快速Format代码。

选中方法名或者变量名 然后使用Shift+Alt+J 快捷键 快速生成注释代码。（注意是否类似搜狗的快捷键导致的冲突后操作未能生效，去掉搜狗快捷键或者更改Eclipse的快捷键）

## 空行

相对独立的程序块之间、变量说明之后必须加空行。

示例：如下例子不符合规范。

//查询界面录入信息值

pkVin = atx.getStringValue("S\_PK\_FULL\_VEC\_ON\_VIN","");//计划过点vin

frameNo = atx.getStringValue("S\_PK\_FULL\_VEC\_ON\_FRAME\_NO");

balaNo = atx.getStringValue("S\_PK\_FULL\_VEC\_ON\_BALA\_NO");

if (PpOrderDao.getPpVinCountByFrameNo(con, frameNo)) {

ErrorHandler.displayError(atx, "此车架编号已进行过绑定，请确认！");

}

应如下书写

//查询界面录入信息值

pkVin = atx.getStringValue("S\_PK\_FULL\_VEC\_ON\_VIN","");//计划过点vin

frameNo = atx.getStringValue("S\_PK\_FULL\_VEC\_ON\_FRAME\_NO");

balaNo = atx.getStringValue("S\_PK\_FULL\_VEC\_ON\_BALA\_NO");

if (PpOrderDao.getPpVinCountByFrameNo(con, frameNo)) {

ErrorHandler.displayError(atx, "此车架编号已进行过绑定，请确认！");

}

## 换行

小节中是详细的规范，请先仔细阅读，后续可以直接使用Ctrl+Shift+F 快捷键， 快速Format代码格式化更为方便，并且统一

### 较长的语句（>80字符）要分成多行书写，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要进行适当的缩进，使排版整齐，语句可读。

示例：

if (PpCabPassDao.getPpCabPassByPk(con, pkCabNo, factoryCode,

currentStationCode, dbTime) != null) {

DynaBeanMap currentStation = PpStationDao.queryPpStationForDelete(

con, currentStationCode);// 获取当前采集点信息

ErrorHandler.displayError(atx, "此驾驶室编号已过"

+ currentStation.get("STATION\_NAME") + "！");

}

### 循环、判断等语句中若有较长的表达式或语句，则要进行适应的划分，长表达式要在低优先级操作符处划分新行，操作符放在新行之首。

if (atx.getStringValue("limit") !=null

&& !atx.getStringValue("limit").equals("")) {

pageSize = new Integer(atx.getStringValue("limit"));

}

### 若函数或过程中的参数较长，则要进行适当的划分。

DynaBeanMap currentStation = PpStationDao.queryPpStationForDelete(

con, currentStationCode);// 获取当前采集点信息

### 不允许把多个短语句写在一行中，即一行只写一条语句。

示例：如下例子不符合规范

String factoryCodeForNo = “DYX000”;String factoryCode = “DYX”;

应如下书写

String factoryCodeForNo = “DYX000”;

String factoryCode = “DYX”;

### 大括号

if、for、do、while、case、switch、default等语句自占一行，且if、for、do、while等语句的执行语句部分无论多少都要加括号{}。并且括号单独一行

示例：如下例子不符合规范

if (part != null)

dsRow.setValue(test,part.getDescription();

应如下书写

if (part != null)

{

dsRow.setValue(test,part.getDescription();

}

## 空格

在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前、之后或者前后要加空格；进行非对等操作时，如果是关系密切的立即操作符（如－>），后不应加空格。

小节中是详细的规范，请先仔细阅读，后续可以直接使用Ctrl+Shift+F 快捷键， 快速Format代码格式化更为方便，并且统一

### 在类型强制转化操作符与操作对象间不应有空格。

示例：如下例子不符合规范

y = (double) x + 16.7;

应如下书写

y = (double)x + 16.7;

### 1.1.2. 在条件语句标识符if与其后的括号间应有一个空格。

示例：如下例子不符合规范

if(x < y);

应如下书写

if (x < y);

### 在条件语句中，表达式与括号间不应有空格。

示例：如下例子不符合规范

if ( x < y );

应如下书写

if (x < y);

### 方法名与括号间以及方法的参数与括号间都不应有空格。

示例：如下例子不符合规范

return ( x + y );

应如下书写

return(x + y);

### 赋值运算符前后应各有一个空格，包括=, +=, -=, \*=, /=, %=, >>=, <<=, &=, 和^=。

### 条件运算符前后应各有一个空格，包括?和:。

### 关系运算符前后应各有一个空格，包括<，<=，>，>=，==，和!=；

### 逻辑运算符前后应各有一个空格，包括&& 和||。

### 位运算符前后应各有一个空格，包括&，^，|，<<，和>>。

### **算术运算符前后应各有一个空格，包括\*，/，%，+，和-，不包括++和--。**

**注：若操作符前后有括号，参照规则13。**

### 逗号和分号后应有空格。

### 一元运算符与操作数间不应有空格，-(negation)，+ (value of operand)，++，--，!。

### 在一级运算符前后不应有空格，包括()，[]，.,，和->。

**注：条件判断语句参照规则2、3。**

### 在局部变量声明和代码间应至少有一个空行。

# 注释规范

## 注释量

一般情况下，源程序有效注释量必须在20％以上。

说明：注释的原则是有助于对程序的阅读理解，在该加的地方都加了，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁。

可以选中方法名或者变量名（类名） 然后使用Shift+Alt+J 快捷键 快速生成注释代码，新建文件的时候也会有自动注释的代码，添加相应的描述即可

## 注释规则

### 文件注释

文件注释写入文件头部，包名之前的位置。注释必须列出：版权说明、生成日期、作者、功能、修改日志等，头文件的注释中还应有函数功能简要说明。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*版权：HanThink 汉思信息技术有限公司

\*作者：

\*日期：

\*描述： // 用于详细说明此程序文件完成的主要功能，与其他模块

// 或函数的接口，输出值、取值范围、含义

// 及参数间的控制、顺序、独立或依赖等关系

\*修改历史: // 修改历史记录列表，每条修改记录应包括修改日期、修改

// 者及修改内容简述

\*1. 日期：

作者：

更正内容：

更正原因：

\*2. 日期：

作者：

更正内容：

更正原因：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

package com.hanthink.mes.pp.action;

### 函数注释

函数头部应进行注释，列出：函数的目的/功能、作者、输入参数、输出参数、返回值、调用关系（函数、表）等。

示例：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：

\*作者：

\*描述：

\*参数：

\*返回：

\*异常：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

英文示例：

/\*\*

\* Name:

\* author:

\*Date:

\*/

### 代码注释

1. 边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。
2. 注释的内容要清楚、明了，含义准确，防止注释二义性。

说明：错误的注释不但无益反而有害。

1. 避免在注释中使用缩写，特别是非常用缩写。

说明：在使用缩写时或之前，应对缩写进行必要的说明。

1. 注释应与其描述的代码相近，对代码的注释应放在其上方或右方（对单条语句的注释）相邻位置，不可放在下面，如放于上方则需与其上面的代码用空行隔开。

示例：如下例子不符合规范。

例1：

pkVin = atx.getStringValue("S\_PK\_FULL\_VEC\_ON\_VIN", "");//计划过点vin

frameNo = atx.getStringValue("S\_PK\_FULL\_VEC\_ON\_FRAME\_NO");

balaNo = atx.getStringValue("S\_PK\_FULL\_VEC\_ON\_BALA\_NO");

//查询界面录入信息值

应如下书写

//查询界面录入信息值

pkVin = atx.getStringValue("S\_PK\_FULL\_VEC\_ON\_VIN", "");//计划过点vin

frameNo = atx.getStringValue("S\_PK\_FULL\_VEC\_ON\_FRAME\_NO");

balaNo = atx.getStringValue("S\_PK\_FULL\_VEC\_ON\_BALA\_NO");

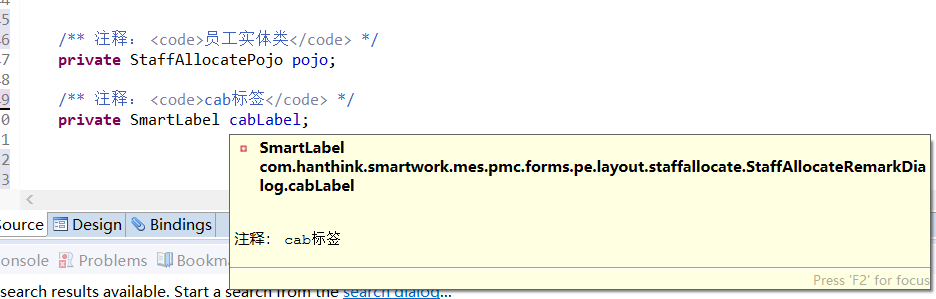
1. 对于所有有物理含义的变量、常量，如果其命名不是充分自注释的，在声明时都必须加以注释，说明其物理含义。变量、常量、宏的注释应放在其上方相邻位置或右方。

示例：

public static final int SORT\_T=20;// VIN号生产

更推荐采用这种注释方式（能直接在Eclispe中光标移动到变量上能看到注释含义），特别是类成员变量：

/\*\* 注释： <code>员工实体类</code> \*/

private StaffAllocatePojo pojo; 

1. 将注释与其上面的代码用空行隔开。

示例：如下例子，显得代码过于紧凑

/\* code one comments \*/

program code one

/\* code two comments \*/

program code two

应如下书写

/\* code one comments \*/

program code one

/\* code two comments \*/

program code two

### 必写注释

变量的定义和分支语句（条件分支、循环语句等）必须编写注释。

说明：这些语句往往是程序实现某一特定功能的关键，对于维护人员来说，良好的注释帮助更好的理解程序，有时甚至优于看设计文档。

示例：

if(AF\_Z14.equals(bindStation)){ // 判断发动机

int engCount = 0;

for(int i = 0 ; i <partbarcode.length; i++){

int indexFirst = partbarcode[i].toString().indexOf("\*", 1);

if(indexFirst <= 1){

continue;

}

}// end for(int i = 0 ; i < partbarcode.length; i++)

if (0 ==engCount){

ErrorHandler.displayError(atx, "当前列表不存在发动机条码！");

}

…

}else if(AF\_Z18.equals(bindStation)){// 判断驾驶室

…

}// end if(AF\_Z14.equals(bindStation))

# 命名规范

## 包的命名

Java包（Package）属于一种特殊情况：它们全都是小写字母，即便中间的单词亦是如此。

为了保障每个Java包命名的唯一性，要求开发人员在自己定义的包的名称之前加上唯一的前缀。由于互联网上的域名称是不会重复的，所以一般采用自己在互联网上的域名称作为自己程序包的唯一前缀。

示例：

com.hanthink.

各个模块的命名：各个模块子功能放在各自模块的下面，不要单独一个包名和各模块组并列，模块缩写可以由各自项目组定义，目前常见的定义如下：

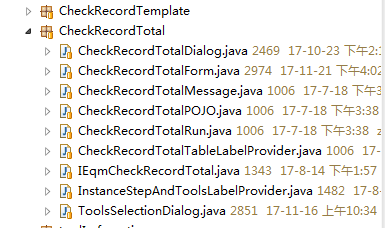
mm-物料管理 pp-计划管理 sm-系统管理 qm-质量管理 mm-制造管理（生产） pub-公共模块 report-报表

## 类的命名

类的名字必须由大写字母开头而单词中的其他字母均为小写，例如Circle；如果类名称由多个单词组成，则每个单词的首字母均应为大写例如TestPage；如果类名称中包含单词缩写，则这个所写词的每个字母均应大写，如XMLExample，还有一点命名技巧就是由于类是设计用来代表对象的，所以在命名类时应尽量选择名词。

1. 功能主界面的类命名： 以描述功能 + 后缀 Form结尾
2. 功能主界面的弹窗类命名： 以描述功能 +后缀 Dialog结尾
3. 实体类即用SQL查询需要填入的类命名： 以功能描述 + 后缀（POJO/Bean/Entity）结尾
4. 使用TableViewer进行解析的的类命名： 以功能描述 +后缀LabelProvider 结尾
5. 用于主界面Form测试的以： 功能描述主界面+后缀Run结尾
6. 用于DS表的字段和表名存放：使用I 开头 + 功能描述 命名
7. 模块化的界面 （主界面各个模块不在一个类中）使用 ：功能描述 +Composite（即模块继承或者使用的组件类型）结尾
8. 操作UI界面的一些类：要求封装在以 主界面+ Util的类名中

下图是命名的java类的例子（做到看到类名知道这个类的作用）:



## 方法的命名

方法的名字的第一个单词应以小写字母作为开头，后面的单词则用大写字母开头，如sendMessge。

## 变量的命名

变量名一般用有意义的名词或名词短语来表示，请不要用下划线。如果只有一个单词，则全部小写，如果有多个单词，则头一个单词小写，其余单词的首字母大写。如int pageNum。

要避免用单个字母来对变量进行命名，除了临时变量和循环变量。

对临时变量和循环变量的命名一般来说：

•by for a byte

•c for a char

•d for a double

•e for an Exception object

•f for a float

•g for a Graphics object

•i, j, k, m, n for integers

•p, q, r, s for String, StringBuffer, or char[] objects

•b for a boolean

•str for String

•sh for short

## 常量的命名

常量的名字应该都使用大写字母，并且指出该常量完整含义。

如果一个常量名称由多个单词组成，则应该用下划线来分割这些单词，如MAX\_VALUE。

## 参数的命名

参数的命名规范和变量的命名规范相同，而且为了避免阅读程序时造成迷惑，请在尽量保证参数名称为一个单词的情况下使参数的命名尽可能明确。

## 数组变量的命名

数组变量名一般用有意义的名词或名词短语来表示，请不要用下划线。如果只有一个单词，则全部小写，如果有多个单词，则头一个单词小写，其余单词的首字母大写。如int [] pageNums，不要使用int pageNums[]。

## 命名要求

1. 命名要清晰、明了，有明确含义，同时使用完整的单词或大家基本可以理解的缩写，避免使人产生误解。

说明：较短的单词可通过去掉“元音”形成缩写；较长的单词可取单词的头几个字母形成缩写；一些单词有大家公认的缩写。

示例：如下单词的缩写能够被大家基本认可。

temp 可缩写为tmp；

flag 可缩写为flg；

statistic 可缩写为stat；

increment 可缩写为inc；

message 可缩写为msg；

1. 命名中若使用特殊约定或缩写，则要有注释说明。

说明：应该在源文件的开始之处，对文件中所使用的缩写或约定，特别是特殊的缩写，进行必要的注释说明。

1. 自己特有的命名风格，要自始至终保持一致，不可来回变化。

说明：个人的命名风格，在符合所在项目组或产品组的命名规则的前提下，才可使用。（即命名规则中没有规定到的地方才可有个人命名风格）。

1. 命名规范必须与所使用的系统风格保持一致，并在同一项目中统一，比如采用UNIX的全小写加下划线的风格或大小写混排的方式，不要使用大小写与下划线混排的方式，用作特殊标识如标识成员变量或全局变量的m\_和g\_，其后加上大小写混排的方式是允许的。

示例：Add\_User不允许，add\_user、AddUser、m\_AddUser允许。

说明：下面是一些在软件中常用的反义词组。

add / remove begin / end create / destroy

insert / delete first / last get / release

increment / decrement put / get

add / delete lock / unlock open / close

min / max old / new start / stop

next / previous source / target

send / receive source / destination

cut / paste up / down show / hide

## 命名禁止

对于变量命名，禁止取单个字符（如i、j、k...），建议除了要有具体含义外，还能表明其变量类型、数据类型等，但i、j、k作局部循环变量是允许的。

说明：变量，尤其是局部变量，如果用单个字符表示，很容易敲错（如i写成j），而编译时又检查不出来，有可能为了这个小小的错误而花费大量的查错时间。

示例：下面所示的局部变量名的定义方法可以借鉴。

int livWidth;

其变量名解释如下：

l 局部变量（Local）（其它：g全局变量（Global）...）

i 数据类型（Interger）

v 变量（Variable）（其它：c常量（Const）...）

Width 变量含义

这样可以防止局部变量与全局变量重名。

# 编码规范

## 可读性

### 优先级

注意运算符的优先级，并用括号明确表达式的操作顺序，避免使用默认优先级。

说明：防止阅读程序时产生误解，防止因默认的优先级与设计思想不符而导致程序出错。

示例：下列语句中的表达式

word = (high << 8) | low (1)

if ((a | b) && (a & c)) (2)

if ((a | b) < (c & d)) (3)

如果书写为

high << 8 | low

a | b && a & c

a | b < c & d

由于

high << 8 | low = ( high << 8) | low，

a | b && a & c = (a | b) && (a & c)，

(1)(2)不会出错，但语句不易理解；

a | b < c & d = a | （b < c）& d，(3)造成了判断条件出错。

### 枚举或者常量

避免使用不易理解的数字，用有意义的标识来替代。涉及物理状态或者含有物理意义的常量，不应直接使用数字，必须用有意义的枚举或宏来代替。

使用枚举的好处：其他人使用函数的时候只能使用此枚举定义的常量而不是其他不可以用的常量。

示例：如下的程序可读性差。

if (trunk.trunkState == 0){

trunk.trunkState = 1;

...

}

应改为如下形式。

final intTRUNK\_IDLE= 0;

final intTRUNK\_BUSY= 1;

if (trunk.trunkState == TRUNK\_IDLE){

trunk.trunkState = TRUNK\_BUSY;

...

}

### 相邻原则

源程序中关系或者逻辑较为紧密的代码应尽可能相邻。

说明：便于程序阅读和查找。

示例：以下代码布局不太合理。

rect.length = 10;

charPoi = str;

rect.width = 5;

若按如下形式书写，可能更清晰一些。

rect.length = 10;

rect.width = 5; // 矩形的长与宽关系较密切，放在一起。

charPoi = str;

### 避免炫技

不建议使用难懂的技巧性很高的语句，除非很有必要时。大家都能快速读懂的代码才是好代码。

说明：高技巧语句不等于高效率的程序，实际上程序的效率关键在于算法。

示例：如下表达式，考虑不周就可能出问题，也较难理解。

statPoi ++ += 1;

++ statPoi += 1;

应分别改为如下：

statPoi += 1;

statPoi++; // 此二语句功能相当于“statPoi ++ += 1; ”

++ statPoi;

statPoi += 1; // 此二语句功能相当于“++statPoi += 1; ”

## JAVA日志打印输出

后台日志打印统一使用LogUtilities输出，不允许使用System.out.println打印。

System.out.println打印只能直接输出到控制台，是单线程输出，当服务器并发量很大时，大量的日志打印会产生输出流堵塞、排队现象。

服务器的显示性能都很底下，控制台的显示效率很低。

### LogUtilities的使用

常用的日志输出级别有error、info、debug，输出级别error > info > debug。当输出级别为debug时, error、info、debug级别的日志都会打印；当输出级别为info时, error、info级别的日志会打印; 当输出级别为error时, 只有error级别的日志会打印。

系统上线初期日志会设置为debug级别，以便跟踪非常详细的日志，但是日志的量会非常大，占用大量磁盘空间。

系统上线稳定后，一般会设置为info级别，跟踪一般的日志信息

所以，在日志输出时按如下规则划分：

a) 以下日志使用logger.error输出

后台捕获Exception后，需要提示的错误信息

b) 以下日志使用logger.info输出

JDBC调用的SQL语句

程序执行过程中，必要的警示信息

c) 以下日志使用logger.debug输出

输出调试信息，如循环中打印的日志信息

### Log日志管理

日志文件标准的方式是按日期生成，由系统管理员定时做清理

### DS表日志记录

a) 对数据库表进行新增操作

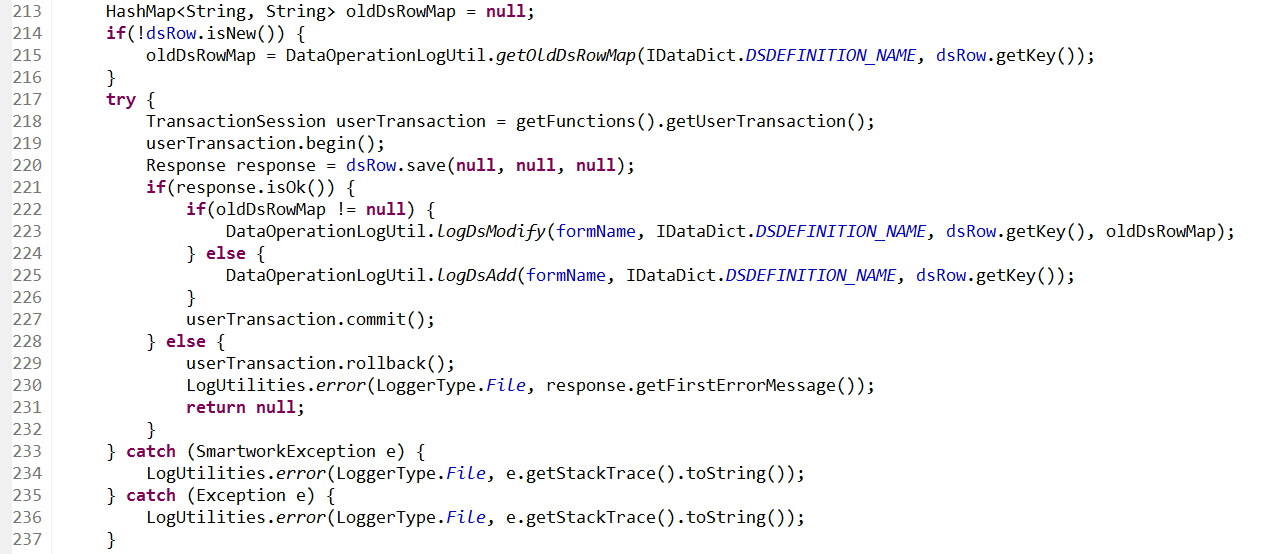
调用DataOperationLogUtil的logDsAdd()方法，参数分别为界面名称、Ds表名、DsRow的Key；

b) 对数据库表进行修改操作

调用DataOperationLogUtil的logDsModify()方法，参数为别为界面名称、Ds表名、DsRow的Key、原记录中的值(DataOperationLogUtil.getOldDsRowMap(Ds表名, DsRow的Key)获取)；

c) 对数据库表进行删除操作

调用DataOperationLogUtil的logDsDelete()方法，参数分别为界面名称、Ds表名、DsRow的Key；

示例如下：

### 对象日志记录

使用ObjectOperationLogUtil类的方法：

/\*\*

\*

\* **@Description** 对象操作日志记录

\* **@param** formName 操作界面名称

\* **@param** obj 操作对象

\* **@param** oldValueMap 对象修改前值(新增则为null)

\* **@param** newValueMap 对象修改后值(删除则为null)

\* **@param** operationType A:新增;D:删除;M:修改

\* ------------------------------------------------------------------------

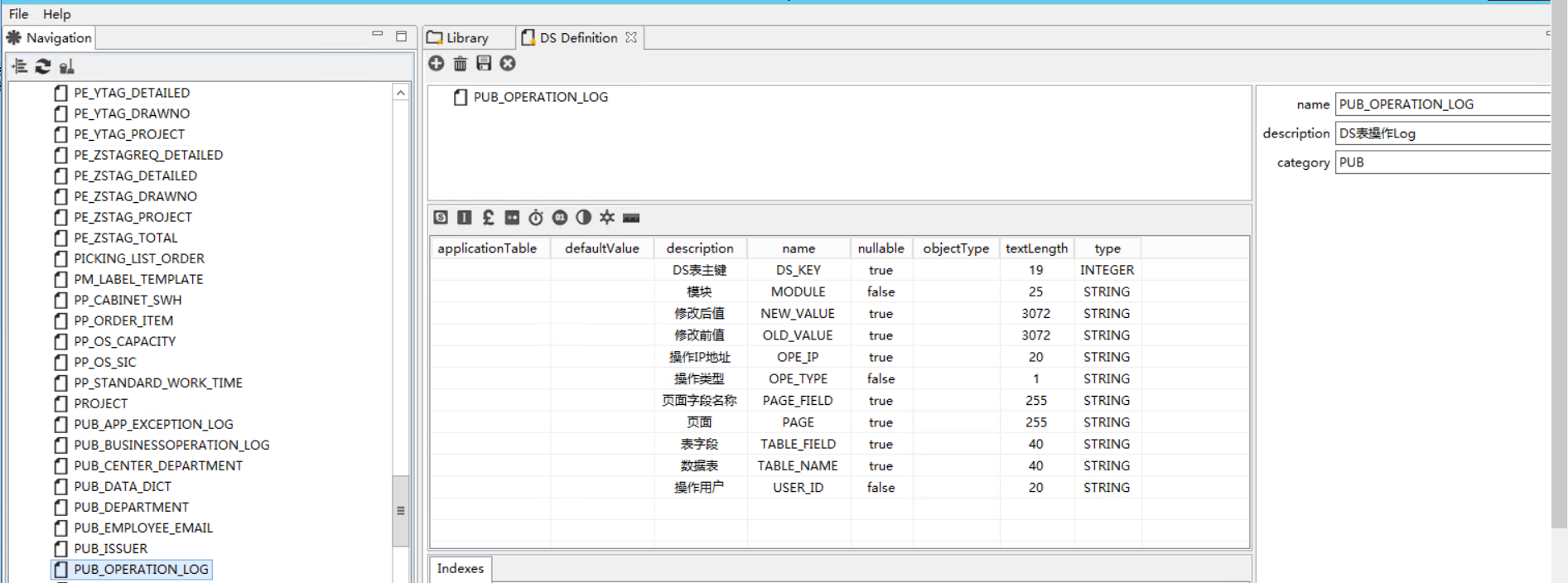
\* Date: Editor: ChangeReasons:

\*

\*/

logObjectModify(String formName, Object obj,HashMap<String, String> oldValueMap, HashMap<String, String> newValueMap,String operationType)

### 操作日志



业务日志BusinessOperationLogUtil类

/\*\*

\*

\* **@Description** 日志新增方法

\* **@param** functions

\* **@param** factory 工厂名

\* **@param** area 车建名

\* **@param** station 站点名

\* **@param** productionLine 生产线名

\* **@param** logType 日志类型 Operation-操作,IF-接口,Procedure-存储过程

\* **@param** operationModule 操作模块

\* **@param** operationCode 操作代码

\* **@param** operationForm 操作界面

\* **@param** operationLevel 操作等级 0,不发送 1,发送

\* **@param** operationContent 操作内容

\* **@param** userName 操作用户

\*

\* **@author**:luoya

\* **@createdate**:下午12:09:59

\* **@ChangeList**

\* --------------------------------------------------------------------------

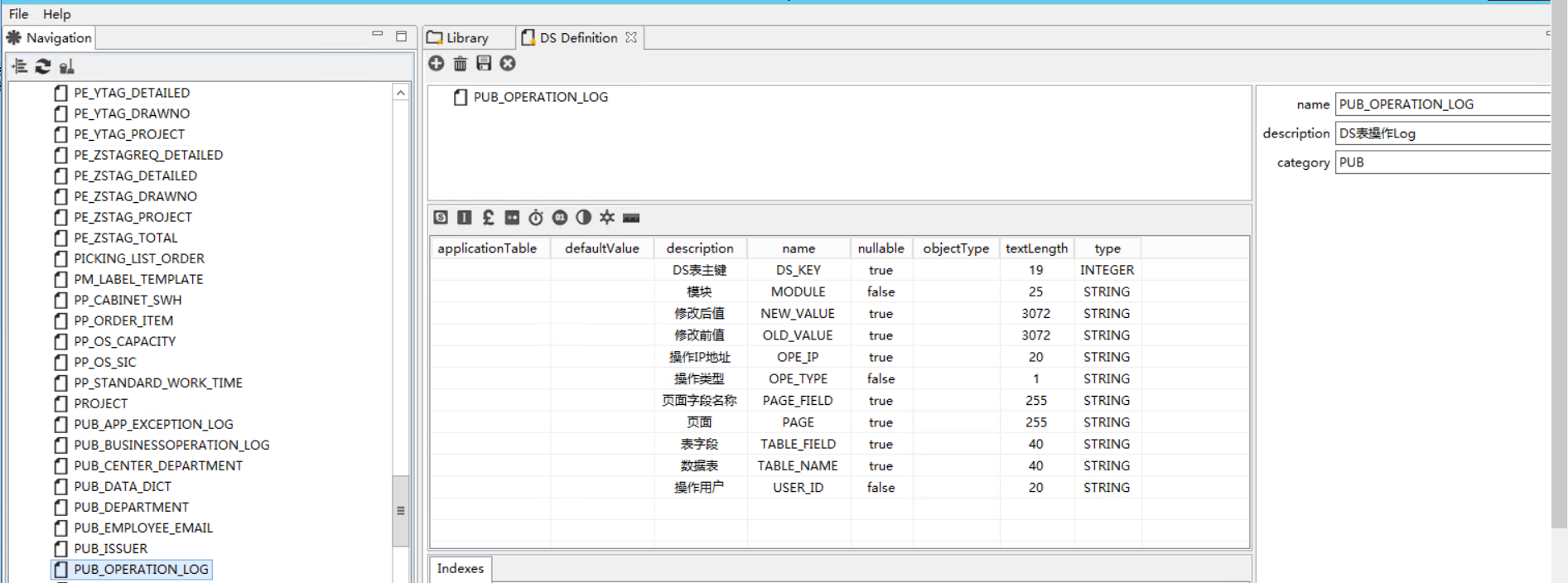
\* Date Editor ChangeReasons

\*

\*/

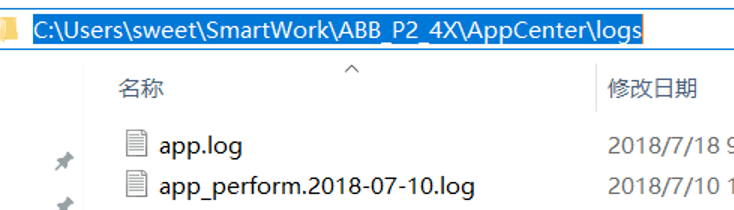
public static void logAdd(Functions functions,String factory,String area,String station,String productionLine,String logType,String operationModule,String operationCode,String operationForm,int operationLevel,String operationContent,String userName)

注意：创建DS表时需要把表的category写出来。



### 日志查看

1. LogUtilities日志记录的位置是根据mes\_logback.xml 配置文件配置的，LoggerType为File和Performance的生成的日志位置如下图



1. DataOperationLogUtil类记录的操作日志是记录到DS\_PUB\_OPERATION\_LOG 表中的。
2. ObjectOperationLogUtil 类记录的对象日志也是记录到DS\_PUB\_OPERATION\_LOG 表中的。

4. BusinessOperationLogUtil类记录到的业务日志 是记录到DS\_PUB\_BUSINESSOPERATION\_LOG 表

## 国际化

### 尽可能的使用国际化

在代码中难免会出现需要用到中文的地方，而国际化可以实现代码中不出现除英文之外的其他语言并且方便日后维护及优化扩展、不同地域对应显示不同语言的作用，所以在编码过程中除注释之外的代码部分尽可能的使用国际化来实现。

### 国际化类命名规范

国际化类的命名规范与普通类的命名规范类似，使命名具有一定含义，一般为模块功能名加Message。

示例：

public class ReworkRecordMessage {

…

}

### 国际化类常量及MessageID命名

常量命名为该常量含义名称加MSG\_ID，包名常量为包名加PACK\_NAME，国际化包中的ID应具有具体意义，有一定的辨识度。

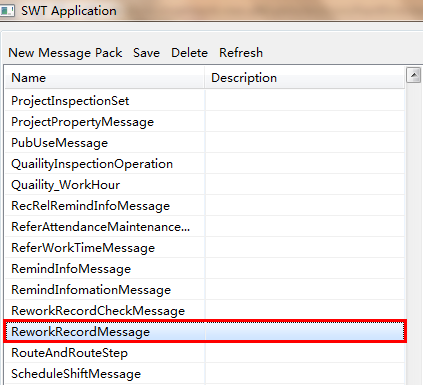
示例：

public final static String DEMO\_MESSAGE\_PACK\_NAME = "ReworkRecordMessage";

public final static String TOOL\_TIP\_QUERY\_MSG\_ID = "tooltip\_query";

### 国际化包名应和国际化类名一致

示例：



### 国际化方法

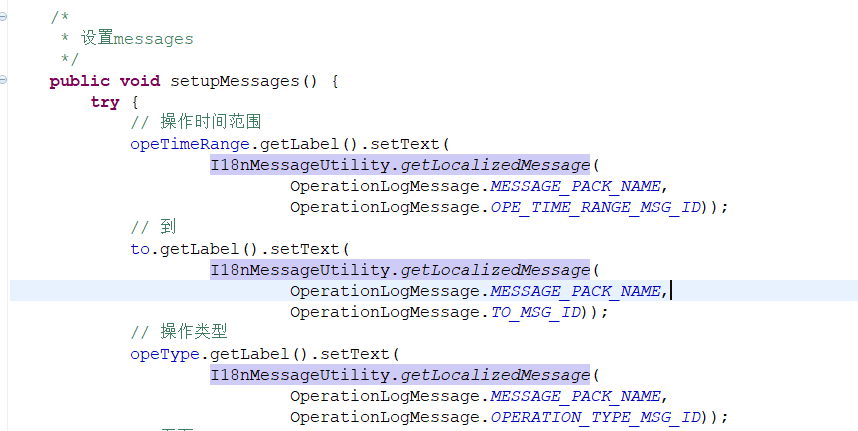
使用I18nMessageUtility类：

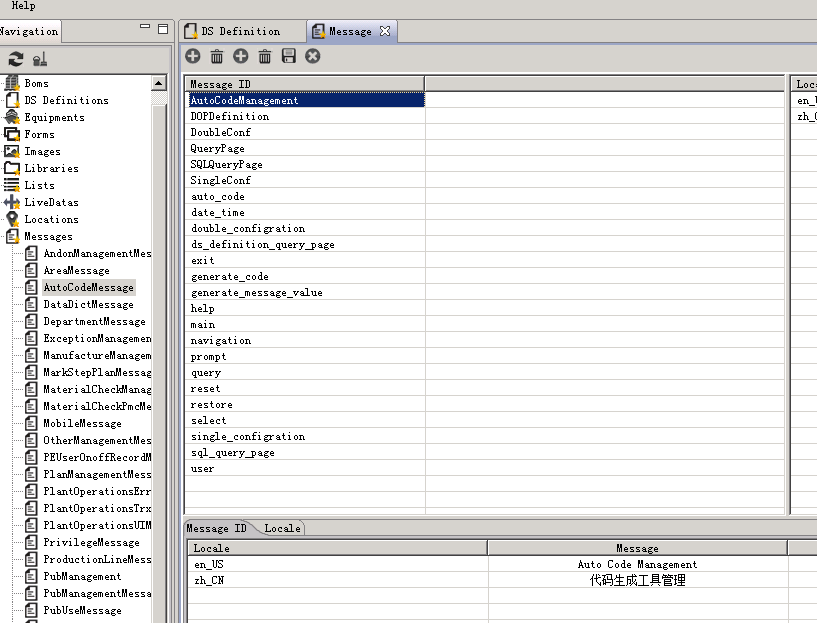
3.X版本的SmartWork中使用I18nMessageUtility.getLocalizedMessage([String](eclipse-javadoc:%E2%98%82=CSRZIC/lib%5C/smartwork-client-all.jar%3Ccom.hanthink.framework.common.utility(I18nMessageUtility.class%E2%98%83I18nMessageUtility~getLocalizedMessage~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String) msgPackName, [String](eclipse-javadoc:%E2%98%82=CSRZIC/lib%5C/smartwork-client-all.jar%3Ccom.hanthink.framework.common.utility(I18nMessageUtility.class%E2%98%83I18nMessageUtility~getLocalizedMessage~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String) msgID) 进行国际化，在运行后，如果没有进行国际化 会自动生成一个msgPackName命名的meassge（即4.3.4的国际化包名和类名自动创建并保持一致），在其下面有一个msgID,此时可以在PD中输入需要国际化的中文和英文，然后保存即可

4.X版本新增了一个方法I18nMessageUtility.getLocalizedMessage([String](eclipse-javadoc:%E2%98%82=ABB_P2_4X/C:%5C/work%5C/eclipse4%5C/plugins%5C/SmartworkDCE%5C/plugins%5C/com.hanthink.smartwork.client_4.2.1.jar%3Ccom.hanthink.framework.client.utility(I18nMessageUtility.class%E2%98%83I18nMessageUtility~getLocalizedMessage~Ljava.lang.String;~Ljava.lang.String;~Ljava.lang.String;%E2%98%82java.lang.String) msgPackName, [String](eclipse-javadoc:%E2%98%82=ABB_P2_4X/C:%5C/work%5C/eclipse4%5C/plugins%5C/SmartworkDCE%5C/plugins%5C/com.hanthink.smartwork.client_4.2.1.jar%3Ccom.hanthink.framework.client.utility(I18nMessageUtility.class%E2%98%83I18nMessageUtility~getLocalizedMessage~Ljava.lang.String;~Ljava.lang.String;~Ljava.lang.String;%E2%98%82java.lang.String) msgID, [String](eclipse-javadoc:%E2%98%82=ABB_P2_4X/C:%5C/work%5C/eclipse4%5C/plugins%5C/SmartworkDCE%5C/plugins%5C/com.hanthink.smartwork.client_4.2.1.jar%3Ccom.hanthink.framework.client.utility(I18nMessageUtility.class%E2%98%83I18nMessageUtility~getLocalizedMessage~Ljava.lang.String;~Ljava.lang.String;~Ljava.lang.String;%E2%98%82java.lang.String) defaultValue)

此方法多了一个defaultValue的默认参数，可以默认填入一个中文或者英文，run类运行后，可以直接根据中文或者英文 生成一个默认的message，此时对应的英文或者中文还需要另外进行填写，默认的message生成后，修改默认值代码不会修改这个默认值。

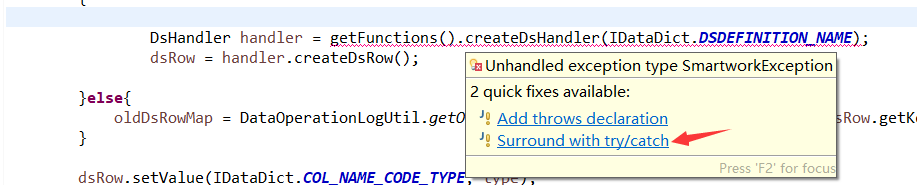
默认的文本标签控件的描述 一般是写在form类中的setupMessages()方法中的。



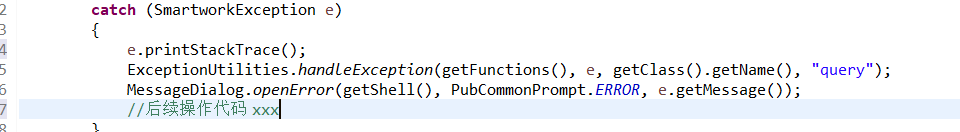


## 异常捕获

添加Code Templates之后，需要抛出异常的部分，点击Surround with try/catch。系统会自动生成需要的异常捕获代码，

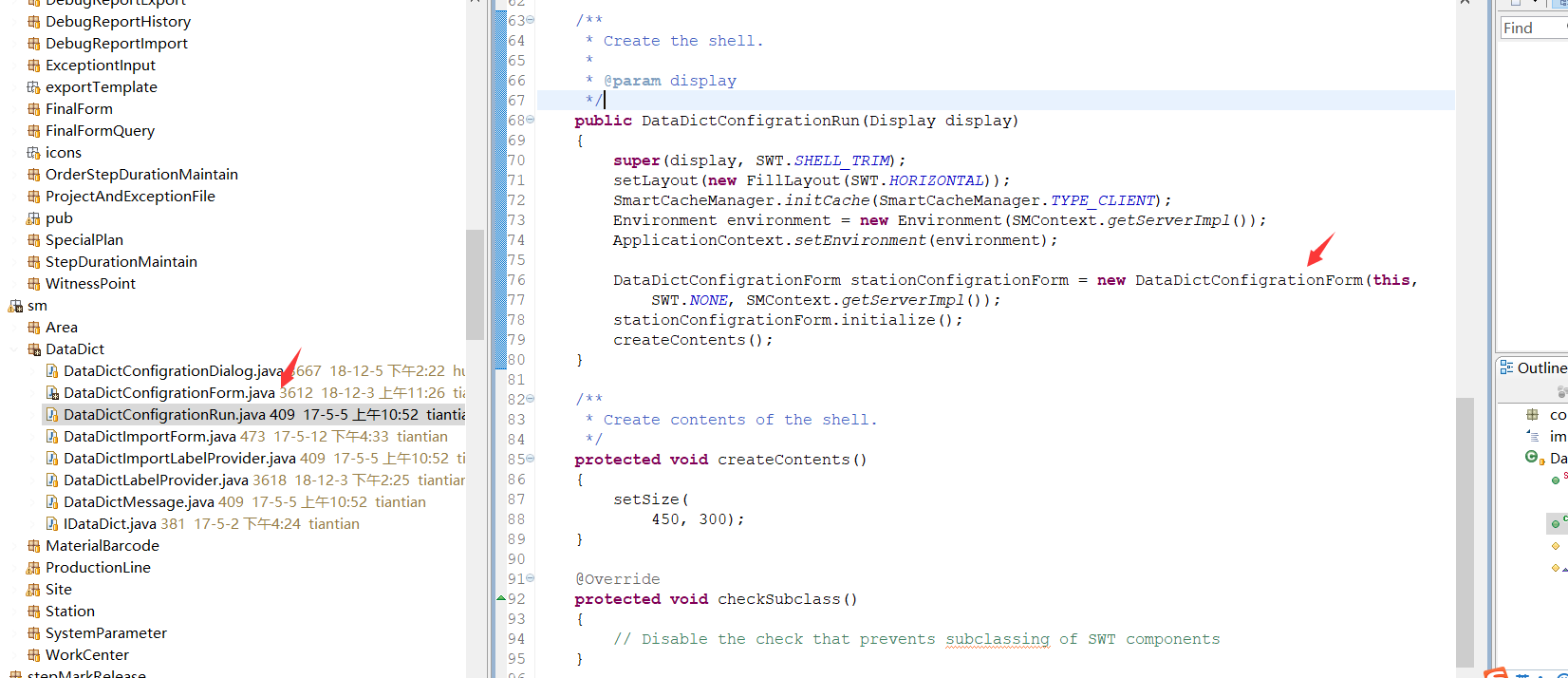


包括记录相关异常到数据库和日志文件。自动生成的代码需要稍加改动，异常捕获后需要操作的代码需要自行编写，另外还需要友好的交互提示代码出错或者异常MessageDialog.openError提示框提示此处发生了错误（或者其他类似方法可以输出到UI界面显示的也可行）。注意只有e.printStackTrace(); 是错误的写法



## 测试代码

SmartWork中界面的测试大都 使用Run类进行测试，核心代码：写需要测试的form类名进行替换后，运行即可测试



## 性能优化

o 避免使用许多子查询来从数据库中获取值，而是使用连接。.

o 避免使用无限循环，因为它倾向于大量利用CPU时间，并使其他进程或应用程序等待。始终确保每个循环都有一个有效的循环终止条件。循环终止条件在某一点上为假，因此循环终止。.

o 当只需要引用对象时，避免重复新建对象。

o SQL查询时注意条件限制，注意不要从数据库获取过多的数据占用到jvm的缓存。

## 类命名唯一

保证在自己类路径指到的任何地方，每个名字都仅对应一个类

为避免编程时遇到麻烦，请保证在自己类路径指到的任何地方，每个名字都仅对应一个类。否则，编译器可能先找到同名的另一个类，并报告出错消息。若怀疑自己碰到了类路径问题，请试试在类路径的每一个起点，搜索一下同名的.class文件。

用合理的设计方案消除“伪功能”。也就是说，假若只需要创建类的一个对象，就不要提前限制自己使用应用程序，并加上一条“只生成其中一个”注释。请考虑将其封装成一个“独生子”的形式。若在主程序里有大量散乱的代码，用于创建自己的对象，请考虑采纳一种创造性的方案，将些代码封装起来。

## 方法设计

应将方法设计成简要的、功能性单元

用它描述和实现一个不连续的类接口部分。理想情况下，方法应简明扼要。若长度很大，可考虑通过某种方式将其分割成较短的几个方法。这样做也便于类内代码的重复使用（有些时候，方法必须非常大，但它们仍应只做同样的一件事情）。

SW方法设计规范：1.业务和UI分离 2.方法参数规范

## 类设计

使类尽可能短小精悍，而且只解决一个特定的问题。下面是对类设计的一些建议：

1、一个复杂的开关语句：考虑采用“多形”机制

2、数量众多的方法涉及到类型差别极大的操作：考虑用几个类来分别实现

3、许多成员变量在特征上有很大的差别：考虑使用几个类

## 私有化

让一切东西都尽可能地“私有”——private。可使库的某一部分“公共化”（一个方法、类或者一个字段等等），就永远不能把它拿出。若强行拿出，就可能破坏其他人现有的代码，使他们不得不重新编写和设计。若只公布自己必须公布的，就可放心大胆地改变其他任何东西。在多线程环境中，隐私是特别重要的一个因素——只有private字段才能在非同步使用的情况下受到保护。

## Import 语句

Import 语句必须详细的写清楚具体的类，尽量避免使用.\* 语句。

示例：如下例子不符合规范

import java.util.\*;

import org.apache.foo.\*;

import org.apache.bar.\*;

应如下书写：

import java.util.ArrayList;

import java.util.Hashtable;

import org.apache.foo.Bar;

import org.apache.bar.Foo;

## “大对象”

对象表达的应该是应用程序的概念，而非应用程序本身

谨惕“巨大对象综合症”。对一些习惯于顺序编程思维、且初涉OOP领域的新手，往往喜欢先写一个顺序执行的程序，再把它嵌入一个或两个巨大的对象里。根据编程原理，对象表达的应该是应用程序的概念，而非应用程序本身。

## 内部类

考虑是否采用内部类。

任何时候只要发现类与类之间结合得非常紧密，就需要考虑是否采用内部类，从而改善编码及维护工作。

## 注释强调

尽可能细致地加上注释，并用javadoc注释文档语法生成自己的程序文档。

规格如下：

/\*\*

\* 方法说明：资金存取总额异常

\* @param ci ComInput 输入参数含义备注信息

\* @return ComOutput 返回结果含义备注信息

\* @throws Exception

\*/

public ComOutput auFundAccessTotal(ComInput ci ) throws Exception{

}

## 对象清除(资源泄露)

进行必要的对象清除。

当客户程序员用完对象以后，若你的类要求进行任何清除工作，可考虑将清除代码置于一个良好定义的方法里，采用类似于cleanup()这样的名字，明确表明自己的用途。除此以外，可在类内放置一个boolean（布尔）标记，指出对象是否已被清除。在类的finalize()方法里，请确定对象已被清除，并已丢弃了从RuntimeException继承的一个类（如果还没有的话），从而指出一个编程错误。在采取象这样的方案之前，请确定finalize()能够在自己的系统中工作（可能需要调用System.runFinalizersOnExit(true)，从而确保这一行为）。

在一个特定的作用域内，若一个对象必须清除（非由垃圾收集机制处理），请采用下述方法：初始化对象；若成功，则立即进入一个含有finally从句的try块，开始清除工作。

若在初始化过程中需要覆盖（取消）finalize()，请记住调用super.finalize()（若Object属于我们的直接超类，则无此必要）。在对finalize()进行覆盖的过程中，对super.finalize()的调用应属于最后一个行动，而不应是第一个行动，这样可确保在需要基础类组件的时候它们依然有效。

SmartWork中

1. MQ的执行需要在shell关闭的时候close()（写一个form类dispose的监听）
2. Timer的使用需要在shell关闭的时候cancel()（写一个form类dispose的监听）
3. 多线程 一些while条件中的变量修改为false 终止循环。
4. 连接数据库， conn 关掉。

## 对象集合

创建大小固定的对象集合时，请将它们传输至一个数组

创建大小固定的对象集合时，请将它们传输至一个数组（若准备从一个方法里返回这个集合，更应如此操作）。这样一来，我们就可享受到数组在编译期进行类型检查的好处。此外，为使用它们，数组的接收者也许并不需要将对象“造型”到数组里。

## Interfaces

尽量使用interfaces，不要使用abstract类。若已知某样东西准备成为一个基础类，那么第一个选择应是将其变成一个interface（接口）。只有在不得不使用方法定义或者成员变量的时候，才需要将其变成一个abstract（抽象）类。接口主要描述了客户希望做什么事情，而一个类则致力于（或允许）具体的实施细节。

## 构造器

1. 抛出构建器内的异常

涉及构建器和异常的时候，通常抛出在构建器中捕获的任何异常——如果它造成了那个对象的创建失败。这样一来，调用者就不会以为那个对象已正确地创建，从而盲目地继续。

1. 在构建器内，只进行那些将对象设为正确状态所需的工作。

尽可能地避免调用其他方法，因为那些方法可能被其他人覆盖或取消，从而在构建过程中产生不可预知的结果。

## 尽量新建类，避免继承

在现成类的基础上创建新类时，请首先选择“新建”或“创作”。只有自己的设计要求必须继承时，才应考虑这方面的问题。若在本来允许新建的场合使用了继承，则整个设计会变得没有必要地复杂。

## 避免使用“魔术数字”

避免使用“魔术数字”，这些数字很难与代码很好地配合。如以后需要修改它，无疑会成为一场噩梦，因为根本不知道“100”到底是指“数组大小”还是“其他全然不同的东西”。所以，我们应创建一个常数，并为其使用具有说服力的描述性名称，并在整个程序中都采用常数标识符。这样可使程序更易理解以及更易维护。

## 警惕“分析瘫痪”

请记住，无论如何都要提前了解整个项目的状况，再去考察其中的细节。由于把握了全局，可快速认识自己未知的一些因素，防止在考察细节的时候陷入“死逻辑”中。

## 警惕“过早优化”

首先让它运行起来，再考虑变得更快——但只有在自己必须这样做，而且经证实在某部分代码中的确存在一个性能瓶颈的时候，才应进行优化。除非用专门的工具分析瓶颈，否则很有可能是在浪费自己的时间。性能提升的隐含代价是自己的代码变得难于理解，而且难于维护。

## 谨惕“巨大对象综合症”

对一些习惯于顺序编程思维、且初涉OOP领域的新手，往往喜欢先写一个顺序执行的程序，再把它嵌入一个或两个巨大的对象里。根据编程原理，对象表达的应该是应用程序的概念，而非应用程序本身。

# 常见的编码错误

## 字符串判断

判断一个字符串变量是否等于某个值

示例：如下例子不符合规范

varStr.equals(“1”)

应如下书写

“1”.equals(varStr)

原因：如果varStr为null，会报空指针错误

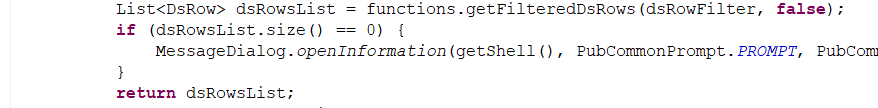
## 数组越界

例：字符串截取前，一定要判断字符串长度，否则会报数组越界

## 慎用全局变量

## 数据库查询

使用从数据库查询出的记录前，一定要判断该记录存在，否则会报空指针。

例如：查询的时候可能获取不到数据，可以给出提示**第二部分 界面规范**

# Layout布局

## AbsoluteLayout

AbsoluteLayout是绝对布局，是SWT界面设计的默认布局方式，采用坐标绝对定位的方式，定义组件的位置和大小。一般不推荐使用绝对布局，布局参数调整麻烦，窗口resize后不能

适应。

## FillLayout

FillLayout是充满式布局，它把组件按一行或一列充满整个容器，并强制组件的大小一致。一般，组件的高度与最高组件相同，宽度与最宽组件相同。FillLayout不能拆行，不能设置边界距离和间距。如果容器中只有一个组件，则该组件会充满整个容器。充满式布局可以在窗口resize后改变组件大小。

1. 构造方法

FillLayout() 创建按一行充满容器的对象。

FillLayout(int type) 创建按指定类型充满容器的对象，指定类型（type）有：SWT.HORIZONTAL 按一行充满容器。SWT.VERTICAL 按一列充满容器。

1. 常用属性

int type指定组件充满容器的类型。type的取值同上。

要将组件按一列充满容器，可以设置type属性，代码如下：

FillLayout filllayout=new FillLayout(); //创建FillLayout对象

Filllayout.type=SWT.VERTICAL; //设置type的值

shell.setLayout(filllayout); //将FillLayout对象用于shell上

new Button(shell,SWT.PUSH).setText("超宽按钮1");//在shell中创建按钮

new Button(shell,SWT.PUSH).setText("按钮2");

new Button(shell,SWT.PUSH).setText("按钮3");

new Button(shell,SWT.PUSH).setText("按钮4");

1. 效果展示



## GridLayout

GridLayout是网格式布局，以网格的方式进行布局，组件可以占用指定的一个或几个网格。网格式布局（GridLayout类）是实用而且功能强大的标准布局，也是较为复杂的一种布局。这种布局把容器分成网格，把组件放置在网格中。GridLayout有很多可配置的属性，和RowLayout一样，也有专用的布局数据类GridData， GridLayout的强大之处在于它可以通过GridData来设置每一个组件的外观形状。GridLayout的构造方法无参数，但可以通过GfidData和设置GridLayout的属性来设置组件的排列及组件的形状和位置。

1. GridLayout的属性

int numColumns：设置容器的列数，组件从左到右按列放置，当组件数大于列数时，下一个组件将自动添加新的一行。默认值为1列。

boolean makeColumnsEqualWidth：强制使列都具有相同的宽度，默认值为false。

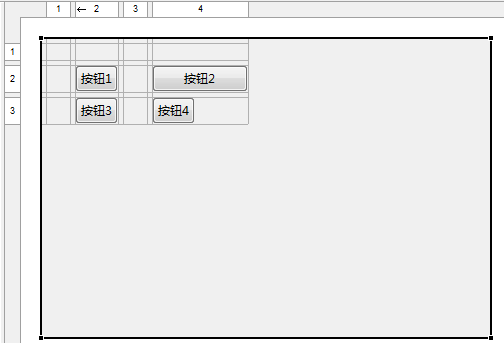
int marginWidth：设置组件与容器边缘的水平距离，默认值为5。

int marginHeight：设置组件与容器边缘的垂直距离，默认值为5。

int horizontalSpacing：设置列与列之间的间隔，默认值为5。

int verticalSpacing：设置行与行之间的间隔，默认值为5。

1. 效果展示



## FormLayout

FormLayout是表格式布局，表格式布局（FormLayout类）是一种非常灵活、精确的布局方式。FormLayout也有专用的布局数据类FormData，此外，还增加了一个FormAttachment类。FormAttachment定义了组件的四边与父容器（Shell、Composite等）的边距，为保证组件在父容器中的相对位置不变，FormAttachment类用不同的构造方法来实现组件的定位，用FormData和FormAttachment配合，可以创建复杂的界面，而且当主窗体大小改变时，组件的相对位置能保持相对不变。FormLayout的构造方法：FormLayout()。

FormLayout的属性：

int marginWidth：设置组件与容器边缘的水平距离，默认值为0。

int marginHeight：设置组件与容器边缘的垂直距离，默认值为0。

# 容器使用

通常，组件构建在容器类中，容器构建在主窗体（shell）中，主窗体也是容器，也就是说，容器不仅可以容纳组件，也可以容纳容器。有了容器，就可以通过它来对组件进行集体操作。例如，容器在界面上移动时，其上的组件也会随着容器移动，容器隐藏，其组件也会被隐藏，容器销毁（dispose），其组件也会被销毁。

## 面板

面板（Composite类）是最常用的容器。主窗体（shell）是面板（Composite）的子类。面板的构造方法格式如下：Composite(Composite parent,int style)第一个参数表示该容器创建在哪个容器上，第二个参数表示容器的式样。Composite的式样一般都是用SWT.NONE，这时Composite在界面是不显示出来的，只是发挥着容器的作用。如果要让容器形成凹陷效果，可以用SWT.BORDER式样。例如，在主窗体中创建一个容器：Composite composite=new Composite(shell,SWT.NONE);

Composite的常用方法：getLayout()：得到布局管理器。getLayoutData()：得到布局数据。getParent()：得到容纳该容器的父容器。getShell()：得到容纳该容器的Shell。layout()：将容器上的组件重新布局，相当于刷新。

FocusableComposite：带有聚焦效果的容器。

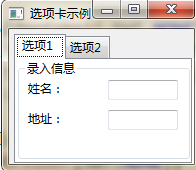
## 分组框

分组框（Group类）是面板（Composite类）的子类，所以两者用法基本相同。主要区别是Group显示有一个方框，且方框线上还可以显示说明文字。

## 选项卡

选项卡包括一个选项卡（TabFolder类）和一个选项页（TabItem类），TabFolder是容器，可以容纳其他容器和组件，但TabItem不是容器，可以把它看成是一个选项标签，TabFolder通过TabItem来对其中的组件进行控制。每一个TabItem用setControl()方法来控制一个界面组件。

如图：



# 控件使用

## 控件使用原则

1. 下拉框和弹出窗使用原则：数据字典类型使用下拉框，主数据使用弹出窗。
2. 数据需要分组，即需要使用Group控件来布局。

## 控件边框

注意边框（控件的border）以美观为主，相邻控件以及控件内部还包含的控件这种布局不能大于2个或者2个以上的边框。

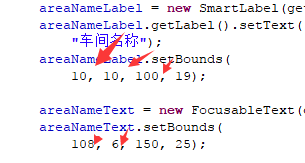
## 控件布局

第一行控件位置推荐，Label（X:10， Y:10），输入控件（X:\*， Y:7），控件行间距推荐：30，例如第一个控件Y：10，则第二个控件Y：40。

表格：FillLayout, GridLayout, BoxLayout, BorderLayout. 表格要求能够自适应界面。界面放大缩小时，可以自动适应大小。

FocusableText 长度默认为150 高度 25

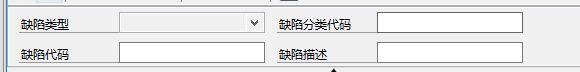
FocusableText 长度默认为100 高度 19



默认高度差为4， 必须横向对齐，并且label后紧挨text （10+100-2 =108），注意-2

上下两行标签高度间隔固定为30 （注：Y坐标差30，即第一行的y坐标如果是10，第二行就是40），查询面板的前后两个标签+文本组合 间隔为10，

效果如下



FocusableCombo 规定类似，以对齐为主。

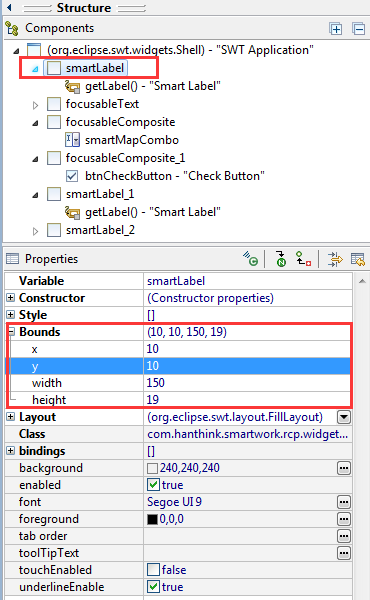
标签的下部距离Composite 距离为（2），一行查询面板至多只能放三个查询条件，基础面板已经有自动扩展，其他自己写的查询面板请注意， 新增面板下面需要留出其他按钮，美观即可。新增面板中两列 控件的Y坐标间距为30，如果低于4个新增的字段，建议直接使用一列即可

## Label标签

Label：文字标签，标准大小推荐：长150，高19。文字左对齐。

SmartLabel：带有下划线的标签，标准大小推荐：长150，高19。文字左对齐。

注：调整大小时，点击Label的容器，即可调整大小。



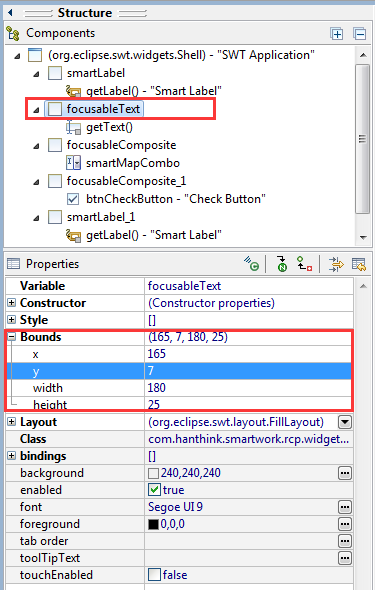
## Text文本框

Text：普通文本输入框，标准大小推荐：长度自定义（默认150，不需要等长），高度25。

FocusableText：具有聚焦效果的输入框。标准大小推荐：长度自定义（默认150，不需要等长），高度25。

注：调整大小时，点击Text的容器，即可调整大小。

styleText:可以输入多行信息的文本输入框。大小根据布局需要调整。



## Combo下拉框

Combo：普通下拉框，标准大小推荐：长度自定义（不需要等长），高度25。

FocusableMapCombo：带有聚焦效果的下拉框，标准大小推荐：长度自定义（不需要等长），高度25。

注：调整大小参考上面的文本框。

## SmartEdit多功能控件

SmartEdit：带有下划线的标签和聚焦效果的输入控件，以及查询按钮。标签和按钮均可设置隐藏。可以设置弹出窗的类名。

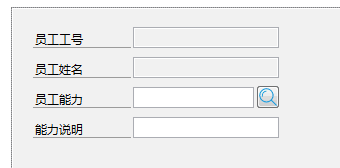
标准大小推荐：Label（长150，高默认），整体不带按钮（长330<不需要等长>，高默认），整体带按钮（长348<不需要等长>，高默认）。

## ToolBar和Button

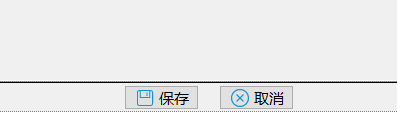
1.ToolBar适用于顶部功能按钮条，单个按钮推荐使用Button

2.布局的时候使用Composite下 包含ToolBar 进行显示，Composite高度为30，ToolBar会以第一个图标的高度进行扩展，第一个图标的image请使用24\*24大小的图片，

使用button,在文本框后使用button进行查询， button大小为(24,24), 不带文字描述，高度比同一行的Text控件高1， 并且与其他列对齐（减少当前列文本的长度）效果如下图



3.其他例如保存 和取消 两个在布局中 也使用Button控件，控件需要加入image图片，此控件大小 默认为 （25,75）， 必要时可以调整，但是需要保证同页面的保存和取消功能按钮大小一致效果如下



另外 Radio Button:单选按钮，标准大小推荐：长度自定义（不需要等长），高度17。

**第三部分 数据库规范**

## 基础面板

项目框架下有封装一些基本的单表和主从表功能面板，尽量继承这些面板进行开发，减少按钮功能代码的开发量，也可以学习代码如何控制，另外可以灵活使用SplitPane进行开发，如下图，效果为一个加强版的SashForm，除了Sashform可以拉动的功能以外，通过中间按钮一键控制收起和扩展以及还原，相关基础面板继承的方法 以及实例，请参考SmartWork的基础培训课程

http://hansi.gensee.com/training/site/v/16082558 http://hansi.gensee.com/training/site/v/99057148 http://hansi.gensee.com/training/site/v/98806802



# PD规范

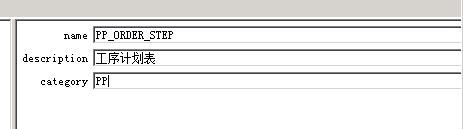
## PROCESS DESIGNER建表规范

### 表名规范

通过process Designer建表后会在表名前自动加DS\_，表名应能够直观表达该表的功能。

并对该表进行简要的描述。表名前缀需要用大模块命名，即表名前加上功能大模块的简写(PP)。并在category中写入模块缩写

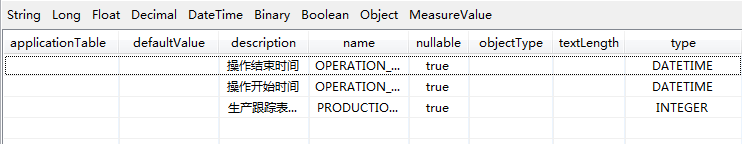
示例：PP\_ORDER\_STEP （注意不能超过30个字符，字符限制）



### 字段命名规范

数据库字段命名应尽可能的表现出该字段的含义，不同单词可用\_链接，并有字段名的意义描述。

示例：



### 字段类型及长度选择

PD中共有9种类型可选择，来满足不同的需求，在类型选择是应根据实际情况选择最合理的数据类型，长度应选择合理，在满足需求的同时不过分浪费存储空间。

9种数据类型分别为：

①.String 字符串

②.Long整数

③.Float小数

④.Decimal数字

⑤.DateTime时间

⑥.Binary二进制

⑦.Boolean布尔

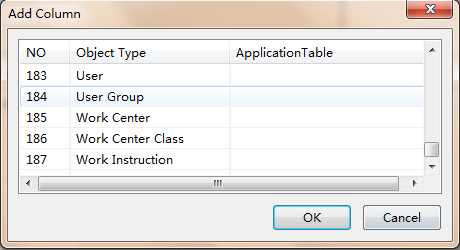
⑧.Object对象

⑨.MeasureValue测量值

### 常用对象

在SmartWork中封装了很多常用对象，在使用过程中尽可能使用对象为存储类型，常用对象如下：

1. User 用户
2. User Group 用户组
3. Work Center 工作中心
4. Station 站点
5. Production Line 生产线
6. DsRow 等



### 培训PD建表规范

在培训过程中的PD建表规范为姓名两个首字母加表名。

例：

BH\_DSROWTEST