**软件开发编码规范**

**(Java、SQL版)**

**版本 1.0**

版 本 历 史

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 修改者 | 日期 | 描述 |
| 1.0 | 李克迪 | 2010.07.20 | 创建java编码规范以及数据库编码规范及命名规则 |
| 1.1 | 李克迪 | 2010.11.11 | 对编码1.0版本规范进行修改，更加实用 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1 概述 3](#_Toc277253235)

[1.1. 内容 3](#_Toc277253236)

[1.2. 编写目的 3](#_Toc277253237)

[1.3. 阅读对象 3](#_Toc277253238)

[2 java编码命名规范 3](#_Toc277253239)

[2.1 Package 的命名 3](#_Toc277253240)

[2.2 Class 的命名 4](#_Toc277253241)

[2.3 变量及方法命名 4](#_Toc277253242)

[2.4 Static Final 变量的命名 4](#_Toc277253243)

[2.5 参数的命名 4](#_Toc277253244)

[2.6 数组的定义及命名 4](#_Toc277253245)

[2.7 方法的参数 4](#_Toc277253246)

[2.8 内部循环变量的命名 5](#_Toc277253247)

[2.9 JavaBean规范 5](#_Toc277253248)

[3 java注释规范 5](#_Toc277253249)

[3.1 单行注释 5](#_Toc277253252)

[3.2 类注释 5](#_Toc277253253)

[3.3 方法注释 6](#_Toc277253254)

[3.4 变量注释 7](#_Toc277253255)

[4 java编码排版规范 7](#_Toc277253256)

[4.1 =间的空格 7](#_Toc277253258)

[4.2 空行 8](#_Toc277253259)

[4.3 换行（试用） 8](#_Toc277253260)

[4.4 缩进 8](#_Toc277253261)

[4.5 声明（试用） 8](#_Toc277253262)

[4.6 括号 9](#_Toc277253263)

[5 java编码格式约定 9](#_Toc277253264)

[6 数据库编码命名规范 11](#_Toc277253265)

[6.1 大小写说明 11](#_Toc277253268)

[6.2 数据库表命名 11](#_Toc277253269)

[6.3 视图命名 11](#_Toc277253270)

[6.4 字段名命名 11](#_Toc277253271)

[7 数据库SQL语句格式 12](#_Toc277253272)

[7.1 大小写说明 12](#_Toc277253274)

[7.2 INSERT INTO 语句 12](#_Toc277253275)

[7.3 UPDATE 语句 12](#_Toc277253276)

[7.4 SELECT 语句 12](#_Toc277253277)

[8 补充说明异常抛出 14](#_Toc277253278)

[8.1 Service层异常的抛出 14](#_Toc277253280)

[8.2 控制层异常的处理 14](#_Toc277253281)

[9 方法修饰说明 15](#_Toc277253282)

# 概述

## 内容

本规范说明书从大方面来说包括：java编码规范、数据库表、字段等的命名、SQL语句排版，以及后面补充的一些说明。

## 编写目的

编码规范对于程序员而言尤为重要，有以下几个原因：

* 一个软件的生命周期中，80%的花费在于维护。
* 几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护。
* 编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码，增加可读性，减少项目组中因为换人而带来的损失。
* 如果你将源码作为产品发布，就需要确任它是否被很好的打包并且清晰无误，一如你已构建的其它任何产品 。

## 阅读对象

本规范说明书阅读对象可以是开发人员、设计人员、测试人员、审查代码人员。

# JAVA编码命名规范

## Package 的命名

Package 的名字应该都是由一个小写单词组成。

例如：

**package** com.somecompany.service.bi.fi;

**package** com.somecompany.service.sox4;

## Class 的命名

Class 的名字必须由大写字母开头而其他字母都小写的单词组成,对于所有标识符，其中包含的所有单词都应紧靠在一起，而且大写中间单词的首字母，一般使用名词命名。

**public** **abstract** **class** AbstractSox4Service {

/\*内容\*/

}

## 变量及方法命名

变量的名字必须用一个小写字母开头。后面的单词用大写字母开头。变量一般使用名词命名，方法名一般使用动词命名，并带有一定的意义，让人一读就懂；再者对于业界认可的单词或固有名词，可直接使用。

例如，变量：userName ,方法名： getUserInformation、addNews

## Static Final 变量的命名

**static final** 变量的名字应该都大写，并且指出完整含义（给出出注释）。

**// Database Configuration PATH**

**public static final** String DB\_CONFIG\_FILE\_PATH ="/cfg/db.properties";

## 参数的命名

参数的名字必须和变量的命名规范一致。

## 数组的定义及命名

数组应该总是用下面的方式来命名：

byte[] buffer; 而不是：byte buffer[];

## 方法的参数

使用有意义的参数命名，如果可能的话，使用和要赋值的字段一样的名字：

setCounter(int size) {

this.size = size;

}

## 内部循环变量的命名

* 规定的变量命名：i, j, k, m, n;
* 长度使用Size；

例如：

**for** (**int** i=1; i<10; i++){ }

**for** (int i=1,size=array.size();i<size;i++){ }

## JavaBean规范

* 一个JavaBean类必须是一个公共类，类修饰符须设置为public；
* 一个JavaBean类必须有一个空的构造函数，必须有一个不带有参数的公用构造器。此构造器也应该通过调用各个特性的设置方法来设置特性的缺省值；
* 一个JavaBean类不应有公共实例变量，类变量都为private或protected，变量命名规范遵上；
* 持有值应该通过一组存取方法（getProperty和setProperty）来访问，对于需要的每个特性，应该有一个带有匹配公用getter 和setter 方法的专用实例变量。

# Java注释规范



## 单行注释

注释要简单明了。

// user’s name used to login this system

String userName = **null**;

## 类注释

位置说明：放在最顶端。

/\*\*

\*

\* 功能描述：写上你的描述，至少能看懂本类是做什么的，有哪些功能

\*

\* @see

\* 与该类相关的类，写出具体的路径：包括完整的包名和类名.java

\*

\* @author (作者) 写上你的姓名

\*

\* @company (开发公司) XXX信息技术有限公司</p>

\*

\* @copyright (版权) 本文件归属XXX信息技术有限公司所有</p>

\*

\* @since (该版本支持的JDK版本)：1.5

\*

\* @version (版本)

\*

\* @date (开发日期) 写上编写日期

\*

\* @modify (修改)

\* <p>第一次修改：时间、修改人；修改内容简介 </p>

\* <p>第二次修改：时间、修改人；修改内容简介 </p>

\* <p>第三次修改：时间、修改人；修改内容简介 </p>

\*

\* @Review (审核人)

\*

\*/

## 方法注释

/\*\*

\* 方法描述

\* @param args array of string arguments

\* @return No return value

\* @exception exceptions No exceptions thrown

\*/

public static void main(String[] args) {

System.out.println(“Hello world !”);

}

如上示例，必须有：

* 方法功能的基本描述
* 每个参数的说明
* 异常的说明
* 返回值的说明

## 变量注释

对于普通变量的注释：

//用户姓名

String username ＝ “xyz”;

对于类变量的注释：

/\* XXXXXX \*/

String username ＝ “xyz”;

# Java编码排版规范



## “=”间的空格

关键词和变量，变量和操作符之间加一个的空格

Options opt1 = null;

## 空行

空行将逻辑相关的代码段分隔开，以提高可读性。下列情况应该总是使用两个空行：

* 一个源文件的两个片段(section)之间
* 类声明和接口声明之间,即从不同来源引用的的包时要用空行隔开加以区分

下列情况应该总是使用一个空行：

* 两个方法之间
* 方法内的局部变量和方法的第一条语句之间
* 块注释或单行注释之前
* 一个方法内的两个逻辑段之间，用以提高可读性

## 换行（试用）

当一个表达式无法容纳在一行内时，可以依据如下一般规则断开之：

* 在一个逗号后面断开
* 在一个操作符前面断开
* 宁可选择较高级别(higher-level)的断开，而非较低级别(lower-level)的断开

## 缩进

对不同级别缩进一个TAB

## 声明（试用）

推荐一行一个声明，因为这样以利于写注释。亦即，

// indentation level

int level;

// size of table

int size;

要优于，

int level, size;

不要将不同类型变量的声明放在同一行，例如：

int foo, fooarray[]; //WRONG!

## 括号

花括号使用如下例

if()空格{

… ;

} else空格{

… ;

}

# java编码格式约定

1. import 语句导入详细的类名，而不是整个目录

例如：

**import** com.somecompany.common.database.DataBase;

**import** com.somecompany.common.exception.DBException;

**import** com.somecompany.service.bi.AbstractBIService;

1. 每个类、成员变量、函数 都必须加注释。注释的格式遵照上面的“注释规范”
2. 已经发布的代码中，不允许出现大块的功能性注释。如果不需要该功能，请直接在代码中删除。
3. 请在已经发布的代码中去掉用于调试的任何代码。
4. 请在已经发布的代码去除所有未使用变量和函数。
5. 每一个成员变量、方法体内变量和成员函数都必须指定其作用域。

关于作用域，在一些不太重视代码质量的人的代码里会发现，一切的变量均在类里的开头一次性定义完，或者更是极为不负责任的创建一大堆的对象，谈到对象的创建，在这里说明一下java的运行机制，举例：

Public class A{}

Public class B extends A{

B (){

A a= New A();

}

}

Public class C extends B{

C (){

A a = New A();

B b =new B();

}

}

这是一段很小的代码，C继承B，B又继承A（事实上A还继承了Object）。

当调用C的时候，在C的构造函数里初始化了A和B，然后程序马上又跑到B里执行，当在B里的时候又发现B的构造函数里初始了A，然后又要执行A的动作，A现在没任何动作，假设A的构造函数里又有一段比较复杂的代码，那么这个程序将花费大量时间在执行。而在C里，我们也没有执行具体的任务，仅仅是进入到C的状态就做了那么多的事情。所以，在一般情况下，我们都不建议一次性做数据的初始化(除非系统非常强烈的要求)，我们建议以实际情况进行分析，合理的安排作用域是有好处的。这样可以很好的维护，也可以提高不少的性能问题，同时也可以排除过多的定义而导致混杂不清的错误。

同样，在方法体内，依然存在局部多用域问题，平时要小心、认真为妙。

1. 为避免虚拟机差异，在声明类的成员变量时，如果不指定其值，应该让其置空；

private Datastore codeDs = null;

...

if(condition){

codeDs=value;

}

If(codeDs==null){

//Do something

}

1. 通常情况下。应该使用日志组件（ 如：log4j）来输出信息、而不是直接使用 System.out；
2. 在构造SQL 时，短小的sql语句尽量使用 “+”，而长度较大的SQL使用StringBuffer 的append 方法；

字符串拼接两种方法：+拼接，StringBuffer拼接

1. 方法内需要打印信息，请直接使用日志组件（如：log.info(……)）方法输出；

# 数据库编码命名规范



## 大小写说明

有关数据库的命名都是用大写，缩写时要取单词的辅音，且单词要有相应意义。

## 数据库表命名

例如：BI\_FIXED\_ASSET (固定资产规模趋势表)

## 视图命名

任何视图必须以V\_开头

例如：V\_ BI \_USER (固定资产规模趋势视图)

## 字段名命名

* 以英文名命名
* 对于多个单词组合的情况，以“\_”分隔
* 单词长度大的使用标准简称
* 字段名应该在20字母以内
* 字段不使用别名

示例：USER\_NAME（用户名）

　　　USER\_PWD（用户密码）

# 数据库SQL语句格式



## 大小写说明

SQL语句必须全部用大写字母编写。

## INSERT INTO 语句

INSERT INTO 表名(字段1, 字段2, 字段3)

VALUES(值1, 值2, 值3)

注意要求：

* 第一行为：INSERT INTO 表(表字段)
* 第二行为：VALUES(字段所对应的值)
* “,”后请打一个空格

## UPDATE 语句

UPDATE 表名

SET 字段1 = 值1, 字段2 = 值2, 字段3 = 值3

WHERE

条件1

AND(OR) 条件2

AND(OR) 条件3

注意要求：

* 第一行 ：UPDATE 表名
* SET设置字段值（注：如果太长，请换行）
* “=”两头请都打空格
* “,”后请打一个空格
* 若带条件，单独一行写WHERE
* 空四个空格符，写上第一个条件
* 若带多个条件，换行，敲两个空格，写AND 条件N
* 一行只写一个 AND 条件

## SELECT 语句

SELECT

字段1，

字段2,

字段3

FROM 表1, 表2, 表3

WHERE

条件1

AND(OR) 条件2

AND(OR) 条件3

ORDER BY 排列字段

GROUP BY 分组字段 HAVING 过滤条件

UNION[ALL]

SELECT ……

或者(SQL标准的关联查询写法)：

SELECT

字段1，

字段2

字段3

FROM 表1

INNER(LEFT、RIGHT 、FULL) JOIN 表1 ON 条件

INNER(LEFT、RIGHT 、FULL) JOIN 表2 ON 条件

CROSS JION 表3 (注：交叉查询是不带 ON 条件的)

WHERE 条件1

AND(OR) 条件2

AND(OR) 条件3

ORDER BY 排列字段

GROUP BY 分组字段 HAVING 过滤条件

UNION[ALL]

SELECT ……

注意要求：

请注意以上的两种格式，该换行的请换行、该空格的请空格！建议采用第二种格式写法(相关的链接查询请正确的理解)

知识提醒：

INNER JOIN 表ON 条件

同等于oracle中的 T1.A = T2.A

LEFT JOIN 表1 ON 条件

同等于oracle中的 T1.A = T2.A(+)

RIGHT JOIN 表1 ON 条件

同等于oracle中的 (+)T1.A = T2.A

# 补充说明异常抛出



## Service层异常的抛出

* 公开外部访问的方法(public)

必须：throws DBException,ServiceException

方法体必须为如下格式的try{ ……} catch

**try**{

//当人为抛出异常时：

**throw** **new** ServiceException("自定义抛出异常信息定义！");

}**catch**(Exception e) {

**throw** **new** ServiceException("service err:"+e.getMessage());

}

* 不公开访问的方法(private)

只有在本类被使用，必须设置为private，此时异常抛出：

1、如果跟数据库相关的，则：

必须：throws DBException, Exception方法体不需要再try{ ……} catch。

2、如果跟数据库无关的操作，则：

必须：throws Exception方法体不需要再try{ ……} catch。

* 单个数据库操作的方法

查询、新增、编辑、删除的单个操作，一律调用

DataBase.*方法*(sqlQuery)。

* 含多个数据库操作的方法

包括(查询、新增、编辑、删除的混合操作)的方法内，一律调用

DataBase.*方法*(sqlQuery, connection)带connection的方法，并且方法结束后一定关闭数据库连接:

**finally** {

DataBase.*closeConnection*(connection);

}

* Service层的方法调用数据库入口说明

调用数据库入口一律以DataBase.方法，不能直接使用这样访问：

AbstractDatabase.方法。

## 控制层异常的处理

捕获：DBException,ServiceException, 或EJBException

# 方法修饰说明

* 公开外部访问的方法——public
* 不公开访问的方法——private
* Service层内的方法一律不允许适用static来修饰

原因：

* + - * 1. static修饰的方法实际上是全局方法，只要是public，在任何地方均能被访问，并且程序一被访问，在内存中已经存在。
        2. 一般复用性极强方法才使用static修饰，如我们的DataBase类，逻辑性强、调用穿插复杂的方法不需要用static修饰，因为不够灵活。
        3. 被static修饰的方法和非static修饰的方法的生命周期不一样。