JAVA软件开发规范

V1.0.1

西安乐食智能餐具有限公司

软件部

2014年5月

修改历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 提交人 | 日期 | 备注（说明修改原因、内容、审核结论等） |
| V1.0.0 | 蔺 博 | 2014.05.19 | 编写“开发编码命名规范.doc” |
|  | 王小龙 | 2014.05.19 | 编写“LESHI项目\_JAVA编码规范.doc” |
| V1.0.1 | 陈威桦 | 2014.05.23 | 合并“开发编码命名规范.doc”和“LESHI项目\_JAVA编码规范.doc”，命名为JAVA软件开发规范 |
|  |  | 2014.05.24 | 1、修改数据库设计规范中表字段命名规则，如果字段有多个单词组成不需要加上“\_”，直接拼接即可  2、修改注释说明 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[修改历史记录 1](#_Toc388708407)

[目录 i](#_Toc388708408)

[1. 概述 1](#_Toc388708409)

[1.1. 为什么需要软件开发规范 1](#_Toc388708410)

[1.2. 预期读者 1](#_Toc388708411)

[1.3. 适用范围 1](#_Toc388708412)

[2. 命名规范 1](#_Toc388708413)

[2.1. 包命名规则 1](#_Toc388708414)

[2.1.1. 与业务相关的包命名 1](#_Toc388708415)

[2.1.2. 与业务系统无关的、可公用的包命名 2](#_Toc388708416)

[2.1.3. 相关工具包 2](#_Toc388708417)

[2.2. 类、接口命名规则 2](#_Toc388708418)

[2.3. 方法命名规则 2](#_Toc388708419)

[2.4. 变量命名规则 3](#_Toc388708420)

[2.5. 常量命名规则 3](#_Toc388708421)

[3. 代码书写规范 4](#_Toc388708422)

[3.1. 返回值 4](#_Toc388708423)

[3.2. 表达式 4](#_Toc388708424)

[3.3. 语句 5](#_Toc388708425)

[3.3.1. if 5](#_Toc388708426)

[3.3.2. for 6](#_Toc388708427)

[3.3.3. while 7](#_Toc388708428)

[3.3.4. switch 7](#_Toc388708429)

[3.3.5. try-catch 8](#_Toc388708430)

[3.4. 注释 9](#_Toc388708431)

[3.4.1. 类说明注释 10](#_Toc388708432)

[3.4.2. 方法说明注释 10](#_Toc388708433)

[3.4.3. 体内代码注释 11](#_Toc388708434)

[3.4.4. 行注释 11](#_Toc388708435)

[3.5. 空白 11](#_Toc388708436)

[3.5.1. 空行 11](#_Toc388708437)

[3.5.2. 空格 12](#_Toc388708438)

[4. 数据库设计规范 12](#_Toc388708439)

[4.1. 表名命名规则 12](#_Toc388708440)

[4.2. 表字段名命名规则 12](#_Toc388708441)

[4.3. 索引命名规则 12](#_Toc388708442)

[4.4. 主键、外键命名规则 13](#_Toc388708443)

[4.5. 数据库规范化设计基本要求 13](#_Toc388708444)

[4.6. 数据库设计规范 13](#_Toc388708445)

[5. 编码安全 15](#_Toc388708446)

# 概述

## 为什么需要软件开发规范

软件开发规范对编程人员相当重要，有如下原因：

* 一个软件的生命周期中，80％的时间用于维护
* 维护工作不太可能始终由一个人来完成
* 编码规范可以改善代码的可读性，便于自己和他人理解
* 编码规范可以降低代码出错的可能性

基于以上原因，为了给自己和同事带来便利，为了提高代码质量，编程人员在编码中必须要遵守软件开发规范。

## 预期读者

软件部全体成员。

## 适用范围

适用于软件部所有基于JAVA开发的项目。

# 命名规范

## 包命名规则

软件部将基于JAVA软件开发中产生的包分为两类，一是与各业务系统相关的包，一是与业务系统无关的、可公用的包。它们的命名规则除要遵守“包名应全部是小写字母，包名中不能出现下划线，并且第一个字母不能是数字”的规则。

### 与业务相关的包命名

与业务系统相关的包命名格式为：

com.leshi.<projectname>.<modulename>

其中： <projectname>为项目英文简称或缩写；<modulename>为模块英文名称或简称，如果无细分模块的话可省略模块名。

### 与业务系统无关的、可公用的包命名

通用包命名格式为：

com.leshi.common.<modulename>//所有项目通用

com.leshi.<projectname>.common//单个项目内各模块通用

### 相关工具包

通用包命名格式为：

com.leshi.utils.<modulename>//所有项目通用

com.leshi.<projectname>.utils//单个项目内各模块通用

## 类、接口命名规则

类和接口的名称应是一个名词，采用大小写混和的方式，如果使用拼音则应是全拼，所有单词都应紧靠在一起，其中每个单词的首字母应大写，接口名称应以大写I结尾。例如：

class User；

interface CommonConnectionI；

class BuMen;//此处为部门全拼

每个类定义要前必须加类的说明。

## 方法命名规则

方法名应是一个动词或动名结构，采用大小写混和的方式，其中第一个单词的首字母用小写，其后单词的首字母大写。例如：

Xxxxx.getBmList()；

每个方法前必须加说明包括：参数说明、返回值说明、异常说明。如果方法名实在是太长可以对变量名缩写，但是必须添加相应的说明。

## 变量命名规则

变量命名一般采用大小写混和的方式，第一个单词的首字母小写，其后单词的首字母大写，变量名一般不要用下划线或美元符号开头。变量名应简短且有意义，即，能够指出其用途。除非是一次性的临时变量，应尽量避免单个字符的变量名。

（1）类的实例对象定义如下：

Person person;

（2）同一个类的多个对象可以采用一下定义方式：

Person person1；

Person person2；

（3）集合类的实例命名使用集合包含元素的英文名称的复数表示，例如：

Vector persons;

（4）如果变量名实在是太长可以对变量名缩写，但是必须在类说明或方法说明部分（视缩写的范围而定）进行说明。

（5）数组的声明要用"int[] packets"的形式，而不要用"int packets[]"。

## 常量命名规则

类常量和ANSI常量的命名应全部用大写，单词间用下划线隔开。例如：

final static int MIN\_WIDTH = 4；

final static int MAX\_WIDTH = 99；

# 代码书写规范

类的方法的代码行数不能过长，尽量控制在100行（90%），长的方法要拆分成私有函数。

## 返回值

在一般情况下，方法返回值不应返回null。而是尽量使用异常代替返回null。如果在特殊情况必须返回null, 必须在方法说明中加以特别说明，如使用“特别注意”等字样。例如：从一个集合类实例中提取一个对象，因为有些集合类实例是允许null作为键或值的，这个时候用异常取代返回null就不合适了。

如果方法的返回值是集合类对象，而且返回的集合对象不包含任何元素时，则应返回0长度或0大小的集合对象。不能返回null。

## 表达式

1. 所有的算术、逻辑表达式的每一项运算都需要加圆括号，避免使用java语言的运算符优先级，例如：

(2 \*(x + y))/(1 - x)；

((n > 1)?(n - 1):(n = 1))

result =（result && (lastOperand > nextOperand)）；

1. 二元算术运算符（除去“/”）、二元逻辑元素符、赋值运算符，既“+、-、\*、%、+=、-=、\*=、/=、%=、>、<、 ==、 >=、<=、 =”等符号左右两边要加空格，例如：

if(lastOperand >= lastOperand)

1. 参数说明部分的逗号“,”和for语句循环说明部分的分号“;”之前不需要留空格，之后需要留空格。如：

Calculator.add(int a, int b);

for(int i = 0; i < 100; i ++);

## 语句

语句包括：

* 1. 条件语句，如if语句、switch语句；
  2. 循环语句，如while语句、for语句。
  3. 异常捕获与处理语句，如try-catch语句。

这些语句的参数说明、条件说明和循环控制都放在圆括号内。如果不是特别长，应尽量放在同一行内。

同时注意，参数说明、条件说明和循环控制的结束圆括号“)”与开始花括号“{”之间留一个空格。

### if

if-else语句有如下几种格式：

|  |
| --- |
| if(condition) {  statements;  }  if(condition) {  statements;  } else {  statements;  }  if(condition) {  statements;  } else if(condition) {  statements;  } else {  statements;  }  //注意：if语句中的内容必须用{}括号括起来  if(condition) //不允许  statements; |

### for

for语句有如下格式：

|  |
| --- |
| //正常for语句  for(initialization; condition; update) {  statements;  }  //所有的操作都在initialization; condition; update中完成  for(initialization; condition; update); |

### while

while语句有如下格式：

|  |
| --- |
| //正常while语句  while(condition) {  statements;  }  //所有的操作都在condition中完成  while(condition); |

一个do-while语句有如下格式：

|  |
| --- |
| do {  statements;  } while(condition); |

### switch

一个switch语句有如下格式：

|  |
| --- |
| switch (condition) {  case ABC:  statements;  /\* 不结束，继续下一个语句 \*/  case DEF:  statements;  break;  case XYZ:  statements;  return; //返回，不需要break  default:  statements;  break;  } |

### try-catch

try-catch语句有如下格式：

|  |
| --- |
| //正常try语句  try {  statements;  } catch (ExceptionClass e) {  statements;  }  //带finally的try语句  try {  statements;  } catch (ExceptionClass e) {  statements;  } finally {  statements;  }  /\*  \* 对于需要显示释放资源的对象，为了避免资源泄漏，  \* 必须要放入try-finally语句中释放  \*/  SomeObj so = null;  try{  so = getObj();  so.doSomething();  }finally{  if(so != null){  so.close();  }  } |

## 注释

注释是软件可读性的具体体现。程序注释量一般占程序编码量的20%，软件工程要求不少于20%。四种必要的注释包括类说明注释、方法说明注释、体内代码注释和行注释。

### 类说明注释

注释一般位于package/import语句之后，class描述之前。要求至少写出名称、描述、创建时间和作者等内容。例如：

/\*\*

\* 名称: WriteOperateLog

\* 描述: 记录用户操作日志

\* 最近修改时间:2014-05-24

\* @since 2014-05-24

\* @author 陈威桦

\*/

### 方法说明注释

对几乎每个方法都应有适当的说明，位于方法声明之前，包括：说明，参数说明、异常说明、返回值说明和特别说明等。例如：

/\*\*

\* 名称：writeLogInfoBefore

\* 描述：记录接收用户请求但没有处理时相关的日志信息

\* @param joinPoint JoinPoint 目标类连接点对象

\* @return void

\* @since 2014-05-24

\* @author 陈威桦

\* @变更记录 2014-05-24 15:00 by 陈威桦

\*/

### 体内代码注释

体(方法体、代码块体、静态代码块体等)内的代码按照功能分成多个虚拟的功能块，每个块以块注释“/\* xxx \*/”注释开始，以空行结束；例如：

/\*\*是否超级管理员\*\*/

private Boolean isadmin = false;

if(null!=curOuId&&!curOuId. equals (“”))

{

/\*组织机构ID不为空时 \*/

curOuId = ‘000000’;

空行

}

空行

if(curOuId==null)

{

/\*组织机构ID为空时 \*/

}

### 行注释

行注释“//”仅用于调试注释，在程序稳定之后，行注释必须被删除，以免影响程序的可读性。

## 空白

### 空行

空行将逻辑相关的代码段分割开，以提高代码的可读性。

* 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
* 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

### 空格

对空格的使用不做要求，以风格统一、清晰易读为准。

# 数据库设计规范

## 表名命名规则

数据库表分为系统表和业务表，系统表的命名以sys开头，业务表的命名以ls开头，名词的复数形式且都为小写，如ls\_user,ls\_categories,ls\_friends等，如果表名由几个单词组成，则单词间用下划线("\_")分割，如ls\_subscribed\_pois,ls\_poi\_categories等表名尽量用全名，表名限制在30个字符内。当表的全名超过30字符时，可用缩写来减少表名的长度，如description --> desc；information --> info；address --> addr等。

## 表字段名命名规则

字段名为小写，字段名为有意义的单词，或单词的缩写，如clientid，postcode等，字段名限制在30个字符内。当字段名超过30字符时，可用缩写来减少字段名的长度，如description --> desc；information --> info；address --> addr等。

## 索引命名规则

索引须按照IDX\_table\_<column>\_<column>,其中<table>是建立索引的表名，<column>是建立索引的字段名，索引名限制在30个字符内。当索引名超过30字符时，可用缩写来减少索引名的长度，如description --> desc；information --> info；address --> addr等 。

## 主键、外键命名规则

主键按照PK\_<table>的规则命名，其中<table>为数据库表名，唯一键按照UK\_<table>\_<column>的规则命名，其中<table>为数据块表名，<column>为字段名，外键按照FK\_<pppp>\_<cccc>\_<nn>的规则命名，其中<pppp>为父表名，<cccc>为子表名，<nn>为序列号。

## 数据库规范化设计基本要求

一、表中应该避免可为空的列。

虽然表中允许空列，但是，空字段是一种比较特殊的数据类型。数据库在处理的时候，需要进行特殊的处理。如此的话，就会增加数据库处理记录的复杂性。当表中有比较多的空字段时，在同等条件下，数据库处理的性能会降低许多。  
 解决方法：一是通过设置默认值的形式，来避免空字段的产生。二是若一张表中，允许为空的列比较多，接近表全部列数的三分之一。而且，这些列在大部分情况下，都是可有可无的。若数据库管理员遇到这种情况，建议另外建立一张副表，以保存这些列。

二、表不应该有重复的值或者列。

三、表中记录应该有一个唯一的标识符。

四、数据库对象要有统一的前缀名。

五、尽量只存储单一实体类型的数据。

六、尽量符合数据库的几个范式。

## 数据库设计规范

所有字段在设计时，除以下数据类型timestamp、image、datetime、smalldatetime、uniqueidentifier、binary、sql\_variant、binary、varbinary外，必须有默认值。字符型的默认值为一个空字符值串’’；数值型的默认值为数值0；逻辑型的默认值为数值0；

其中：系统中所有逻辑型中数值0表示为“假”；数值1表示为“真”。datetime、smalldatetime类型的字段没有默认值，必须为NULL。当字段定义为字符串形时建议使用varchar而不用nvarchar。

* **字段的描述**

数据库中每个字段的描述(Description)如下：

* 表内的每一个值只能被表达一次
* 表内的每一行都应当被唯一的标识
* 表内不应该存储依赖于其他键的非键信息
* 如果字段事实上是与其它表的关键字相关联而未设计为外键引用，需建索引。
* 如果字段与其它表的字段相关联，需建索引。
* **SQL 开发代码规范**

我们规定sql语句编码的时候程序员需要遵循以下规范：

* 所有的关键字的所有字母必须大写。如果一个常量由几个词组成，我们规定两个词之间使用下划线（\_）来划分。表名、列名、视图名或它们的别名必须和它们的定义保持一致。
* 注释必须要规范。
* **其他设计技巧**
* 避免使用触发器，触发器的功能通常可以用其他方式实现。在调试程序时触发器可能成为干扰。假如你确实需要采用触发器，你最好集中对它文档化。
* 使用常用英语(或者其他任何语言)而不要使用编码在创建下拉菜单、列表、报表时最好按照英语名排序。假如需要编码，可以在编码旁附上用户知道的英语。
* 保存常用信息。让一个表专门存放一般数据库信息非常有用。在这个表里存放数据库当前版本、最近检查/修复、关联设计文档的名称、客户等信息。这样可以实现一种简单机制跟踪数据库，当客户抱怨他们的数据库没有达到希望的要求而与你联系时，这样做对非客户机/服务器环境特别有用。
* 包含版本机制。在数据库中引入版本控制机制来确定使用中的数据库的版本。时间一长，用户的需求总是会改变的。最终可能会要求修改数据库结构。把版本信息直接存放到数据库中更为方便。
* 编制文档。对所有的快捷方式、命名规范、限制和函数都要编制文档。采用给表、列、触发器等加注释的数据库工具。对开发、支持和跟踪修改非常有用。对数据库文档化，或者在数据库自身的内部或者单独建立文档。这样，当过了一年多时间后再回过头来做第2 个版本，犯错的机会将大大减少。
* 测试、测试、反复测试。建立或者修订数据库之后，必须用用户新输入的数据测试数据字段。最重要的是，让用户进行测试并且同用户一道保证选择的数据类型满足商业要求。测试需要在把新数据库投入实际服务之前完成。
* 检查设计。在开发期间检查数据库设计的常用技术是通过其所支持的应用程序原型检查数据库。换句话说，针对每一种最终表达数据的原型应用，保证你检查了数据模型并且查看如何取出数据。

# 编码安全

Android开发组可不参考本章内容

遵循下面列出的准则有利于编写更加安全的代码。但是总体来说，这些准则不能对安全性做出任何保证。遵循这些准则可能好的实践，但是即使遵循了这些准则，写出的代码仍然可能是不安全的。风险永远存在，不管在编写代码时是如何的警觉。

这些准则的目标，不是为了保证代码的安全性，而是为了消除若干特定类型攻击带来的风险。遵循这些准则，某些特定类型的攻击将无法实现；但是其它类型的攻击仍然可能成功。因此遵循这些准则仅仅是安全的第一步。当书写可能和非守信链接或混用的代码时，应当仔细的考虑如下准则：

1. **静态字段**

避免使用非final的公共静态变量，应尽可能地避免使用非final公共静态变量，因为无法判断代码有无权限改变这些静态变量的值。

一般地，应谨慎使用可变的静态状态，因为这可能导致设想中应该相互独立的子系统之间发生不曾预期的交互。

1. **缩小作用域**

作为一个惯例，尽可能缩小成员方法和成员变量的作用域。检查包访问权限成员（package-private）能否改成私有成员（private），保护访问成员（protected）可否改成包访问权限成员（package-private）/私有成员（private）等等。

1. **公共方法/字段**

公共变量应当避免使用，访问这些变量时应当通过getter/setter法。在这种方式下，必要时可以增加集中的安全检查。

任何能够访问或修改任何敏感内部状态的公共方法，务必包含安全检查。 参考如下代码段，该代码段中不可信任代码可能修改TimeZone的值：

private static TimeZone  defaultZone = null;

public static synchronized void setDefault(TimeZone zone) {

defaultZone = zone;

}

1. **保护包**

有时需要整体上保护一个包以避免不可信任代码的访问，本节描述了一些防护技术：

防止包注入：如果不可信任代码想要访问类的包保护成员，可能通过在被攻击的包内定义自己的新类用以获取这些成员的访问权的方式。防止这类攻击的方式有两种：

 a. 通过向java.security.properties文件中加入如下文字防止包内被注入恶意类。 ...

package.definition=Package#1 [,Package#2,...,Package#n] ...

当检测到代码试图在包内定义新类时，类装载器的defineClass方法会抛出异常，除非代码被赋予以下权限：

...   RuntimePermission("defineClassInPackage."+package) ...

b. 另一种方式是通过将包放到封闭的JAR（sealed Jar）文件里。 （参看http://java.sun.com/j2se/sdk/1.2/docs/guide/extensions/spec.html） 通过使用这种技巧，代码无法获得扩展包的权限，因此也无须修改java.security.properties文件。

防止包访问：可以通过限制包访问但同时仅赋予特定代码访问权限防止不可信任代码对

包成员的访问。通过向java.security.properties文件中加入如下文字可以达到这一目的： ...

package.access=Package#1 [,Package#2,...,Package#n] ...

当检测到代码试图访问上述包中的类时，类加载器的loadClass方法会抛出异常，除非代码被赋予以下权限：

...  RuntimePermission("defineClassInPackage."+package) ...

1. **尽可能使对象不可变（immutable）**

尽可能使对象不可变。如果对象必须改变，使得它们可以克隆并在方法调用时返回副本。如果方法调用的返回对象是数组、向量或哈希表等，牢记这些对象并非不可变，调用者可以修改这些对象的内容并导致安全漏洞。此外，不可变的对象因为不用上锁所以能够提高并发性。

不要返回包含敏感数据的内部数组引用。

这个不可变惯例的变型，在这儿提出是因为是个常见错误。即使数组中包含不可变的对象比如说是字符串，也要返回一个副本，这样调用者不能修改数组中包含的到底是哪个字符串。在方法调用返回时，返回数据的拷贝而不要返回数组。

1. **不要直接在用户提供的数组里存储**

这是不可变惯例的另一个变型。构造器和方法可以接受对象数组，比如说PubicKey数组，这个数据存储到内部之前应当克隆，并保存克隆后的数据，而不是直接将数组引用赋给同样类型的内部变量。如果缺少这个步骤，在使用了有问题的构造器创建了对象后，用户对外部数组所作的任何修改都将更改对象的内部状态，尽管对象应该是不可变的。

1. **序列化**

对象在序列化后、反序列化之前，都不在Java运行时环境的控制之下，也因此不在Java平台提供的安全控制范围内。

在实现接口Serializable时务必将以下事宜牢记在心：

transient直接引用系统资源的句柄和包含了地址空间相关信息的字段应当使用关键字transient修饰。资源，如文件句柄，如果不被声明为transient，该对象在序列化状态下可能会被修改，从而在被反序列化后获取对资源的不当访问。特定类的序列化/反序列化方法，为了确保反序列化对象不包含违反一些不变量集合的状态，类应该定义自己的反序列化方法并使用接口bjectInputValidation验证这些变量。

如果一个类定义了自己的序列化方法，它就不能向任何DataInput/DataOuput方法传递内部数组。所有的DataInput/DataOuput方法都能被重写。注意默认序列化不会向DataInput/DataOuput字节数组方法暴露私有字节数组字段。

如果Serializable类直接向DataOutput(write(byte [] b))方法传递了一个私有数组，那么黑客可以创建ObjectOutputStream的子类并覆盖write(byte [] b)方法，这样他可以访问并修改私有数组。下面示例说明了这个问题。

示例类：

public class YourClass implements Serializable {

private byte [] internalArray;

....

private synchronized void writeObject(ObjectOutputStream stream) {

...

stream.write(internalArray); ...

 }

 }

黑客代码：

public class HackerObjectOutputStream extends ObjectOutputStream{

public void write (byte [] b) {

Modify b

 }

 } ...

YourClass yc = new YourClass(); ...

HackerObjectOutputStream hoos = new HackerObjectOutputStream();

 hoos.writeObject(yc); 

字节流加密

另一种保护位于虚拟机之外的字节流的方式是对序列化产生的流进行加密。字节流加密可以防止解码和读取被序列化对象的私有状态。如果决定加密，需要管理好密钥，密钥的存储以及密钥交付给反序列化程序的方式，等等。

需要注意的其它事宜

如果不可信任代码在创建对象时受到约束，务必确保不可信任代码在反序列化对象时受到相同的约束。牢记对象反序列化是创建对象的另一途径。

比如说，如果applet创建了frame，在该frame上创建了警告标签。如果该frame被应用程序序列化并被applet反序列化，务必使该frame在反序列化后标有相同的警告标签。

1. **本地方法**

应从以下几个方面检查本地方法：

1. 返回什么
2. 需要什么参数 
3. 是否绕过了安全检查