**项目管理体系文件**

**Java开发编码规范**

编 撰 人：巫于颖

审 核 人：

批 准 人：

批准日期：2010-9-1

保密级别：机密

文档版本：0.0.1

**北京东软信息技术有限公司**

**版本历史**

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**目录**

[第一章 代码开发规范及其指南 1](#_Toc267585157)

[1.1 目的 1](#_Toc267585158)

[1.2 程序内命名规范 1](#_Toc267585159)

[1.3 文件命名规范 2](#_Toc267585160)

[1.4 Java 文件样式 2](#_Toc267585161)

[1.5 代码编写格式 6](#_Toc267585162)

[第二章 程序编写规范方法 8](#_Toc267585163)

[2.1 权限修饰 8](#_Toc267585164)

[2.2 其他规范 8](#_Toc267585165)

[2.3 编程指南 10](#_Toc267585166)

[第三章 其他要求 12](#_Toc267585167)

# 

# 代码开发规范及其指南

## 目的

定义这个规范的目的是让项目中所有的文档都看起来像一个人写的，增加可读性，减少项目组中因为换人而带来的损失。因此要求编码人员尽量遵守（这些规范并不是一定要绝对遵守，但是一定要让程序有良好的可读性）

## 程序内命名规范

* **Package** 的命名 ：Package 的名字应该都是由小写字符串或单词组成。
* **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
* **Class 变量**的命名 ：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。
* **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。
* **参数的命名** ：参数的名字必须和变量的命名规范一致。
* **列表的命名** ：列表应该总是用下面的方式来命名： List<Customer> **list** = **new** ArrayList<Customer>();使用泛型来定义列表的类型。
* **数组的命名** ：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;

}

## 文件命名规范

* 文件名由英文单词组成，每个单词的首字母大写，不要超过4个单词，如WordBook.java。
* 实体类的名称要和数据库的名称对应，并且首字母大写，如有下划线则去掉，如果名称是多个单词的拼接，则按每个单词首字母大写，其余字母小写来命名。
* Java文件的文件名要与程序中的public类名相同。
* Mapper文件要以Mapper作为结尾，如AuthorityMapper.java。
* Mapper文件对应的xml配置文件也要以Mapper作为结尾，并且要与mapper的名称一样，如AuthorityMapper.xml。
* Service接口文件要以I开头，加上对应实体类的名称，以Service作为结尾，如IAuthorityService.java。
* ServiceImpl，也就是Service接口的实现类，以对应实体类名称开头，以ServiceImpl作为结尾，如AuthorityServiceImpl.java。
* 测试文件应尽量以Test作为开头。
* Spring框架的配置文件以application或者spring作为开头，Mybatis框架的配置文件以mybatis开头。
* 新建properties文件时要拼写正确。

## Java 文件样式

所有的 Java(\*.java) 文件都必须遵守如下的样式规则

* **头部**

版权信息

版权信息必须在 java 文件的开头，比如：

|  |
| --- |
| /\*\*  \* Copyright 2018 shuzai.crm.com, Inc. All rights reserved.  \*/ |

其他不需要出现在 javadoc 的信息也可以包含在这里。

* Package/Imports

package 行要在 import 行之前，import 中标准的包名要在本地的包名之前，而且按照字母顺序排列。如果 import 行中包含了同一个包中的不同子目录，则应该用 \* 来处理。

|  |
| --- |
| package com.neuedu.crm.view;  import java.io.\*;  import java.io.File;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.IOException;  import java.io.InputStream;  import java.io.OutputStream; |

这里 java.io.\* 使用来代替InputStream and OutputStream 的。

* **Class**

接下来的是类的注释，一般是用来解释类的相关信息，包括标题、描述、。

|  |
| --- |
| /\*\*  \* <p>Title: Example</p>  \* <p>Description: AuditApplicationExample</p>  \* <p>Copyright: Copyright (c) 2018</p>  \* <p>Date: yyyy-mm-dd</p>  \* @author xxxxxx  \* @version 1.0  \*/ |

接下来是类定义，包含了在不同的行的 extends (父类)和 implements(父接口)

|  |
| --- |
| public class A extends B implements C  public class AuthorityServiceImpl implements IAuthorityService |

* **Class Fields**

接下来是类的成员变量：

|  |
| --- |
| /\*\*  \* 权限表字段  \*/  private List<RoleAuthority> roleauthoritys; |

实体类的字段必须用private来修饰，并且写上注释。protected、private和 package 定义的成员变量如果名字含义明确的话，可以没有注释。

* **存取方法**

接下来是类变量的存取的方法。它只是简单的用来将类的变量赋值获取值的话，可以简单的写在一行上，也可以不写在一行；另外实体类对应数据库字段名有时需要处理为NULL的情况，可以自己写对应处理的getter方法。

|  |
| --- |
| /\*\*  \* Getter and Setter  \*/  public Integer getAuthorityId() {return authorityId; }  public void setAuthorityId(Integer authorityId) {this.authorityId = authorityId; }  public String getDetail() {  if(detail==null) {  return "";  }else {  return detail;  }  }  public void setDetail(String detail) {this.detail = detail;} |

* **构造函数**

接下来是构造函数，每个实体类都必须有构造方法，至少一个无参构造方法，和一个有参数的构造方法，可以按照需要情况添加多个构造方法。

|  |
| --- |
| //无参构造方法  public Authority() {  super();  }  public Authority(Integer authorityId, String authorityMark) {  super();this.authorityId = authorityId;this.authorityMark = authorityMark;} |

* toString 方法

无论如何，每一个实体类都应该定义 toString 方法，这是为了方便测试程序：

|  |
| --- |
| @Override  public String toString() {  return "Authority [authorityId=" + authorityId + ", authorityMark=" + authorityMark + ", sort=" + sort + ", name=" + name + ", roleauthoritys=" + roleauthoritys + "]";} |

* main 方法

如果main(String[]) 方法已经定义了, 那么它应该写在类的底部.

## 代码编写格式

* **代码样式**

代码应该用 windows 的格式，而不是 其他 的（比如：回车变成回车+换行）。

* **文档化**

必须用 javadoc 来为类生成文档。不仅因为它是标准，这也是被各种 java 编译器都认可的方法。

* **缩进**

缩进应该是Tab字符， 在使用不同的源代码管理工具时Tab字符将因为用户设置的不同而扩展为不同的宽度。

* **页宽**

页宽应该设置为80字符. 源代码一般不会超过这个宽度, 并导致无法完整显示, 但这一设置也可以灵活调整. 在任何情况下, 超长的语句应该在一个逗号或者一个操作符后折行. 一条语句折行后, 应该比原来的语句再缩进1个tab字符.

* 程序块

要有大括号，即使只有一句程序

{} 中的语句应该单独作为一行. 例如, 下面的第1行是错误的, 第2行是正确的:

|  |
| --- |
| if (i>0) { i ++ }; // 错误, { 和 } 在同一行  if (i>0) {  i ++  }; // 正确, { 单独作为一行  } 语句永远单独作为一行. |

如果 } 语句应该缩进到与其相对应的 { 那一行相对齐的位置。

* **括号**

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，多个参数之间可以出现空格。下面的例子说明括号和空格的错误及正确使用:

CallProc( AParameter ); // 错误

CallProc(AParameter); // 正确

CallProc(AParameter, BParameter, CParameter); // 正确

不要在语句中使用无意义的括号. 括号只应该为达到某种目的而出现在源代码中。下面的例子说明错误和正确的用法:

if ((I) = 42) { // 错误 - 括号毫无意义

if (I == 42) or (J == 42) then // 正确 - 的确需要括号

* **import**

对于import语句，如果某一个包中引用的类不超过三个（包括三个），尽量不用import xxx.\*;格式。

* **变量**

定义及方法的返回值尽量不使用全包名，如java.lang.String，除非在不同的包中有相同名字的类，必须用全名来区分。对于类中属性的存取要用getter方法和setter方法（getXXX/setXXX，boolean值用isXXX/setXXX），不要直接引用。

# 程序编写规范方法

## 权限修饰

Java中权限修饰符有三个：public,private,protected，另外一种是不写这三种的任何一个，称为（default）.

1. 对于外部类，可以用public或private,但是内部类这四种都可以使用。
2. 如果对某些数据或方法只允许其所在类使用，用private修饰。
3. 如果对某些数据或方法允许本类或同包类使用，用（default）,即不写任何修饰符。
4. 如果对某些数据或方法允许本类，同包类，或非同包子类使用，用protected.

如果对某些数据或方法允许任何类使用，用public修饰。

## 其他规范

* **exit()**

exit 除了在 main 中可以被调用外，其他的地方不应该调用。因为这样做不给任何代码机会来截获退出。一个类似后台服务的程序不应该因为某一个库模块决定了要退出就退出。

* **异常**

关键操作如涉及到数据库操作和多个关键处理步骤时应当采用try,catch语句，以便于捕获异常（各类Exception）和异常处理。如：

FileInputStream is = null;

try {……} catch (FileNotFoundException e) {e.printStackTrace();

} catch (IOException e) {e.printStackTrace();

}

* **垃圾收集**

JAVA使用成熟的后台垃圾收集技术来代替引用计数。但是这样会导致一个问题：必须在使用完对象的实例以后进行清场工作。

除非输出流一出作用域就关闭，非引用计数的程序语言，比如JAVA，是不能自动完成变量的清场工作的。必须像下面一样写，避免不必要的异常：

|  |
| --- |
| finally {try {  is.close();  } catch (IOException e) {  e.printStackTrace();  }  } |

* **final 类**

绝对不要因为性能的原因将类定义为 final 的（除非程序的框架要求）

如果一个类还没有准备好被继承，最好在类文档中注明，而不要将它定义为 final 的。这是因为没有人可以保证会不会由于什么原因需要继承它。

访问类的成员变量

大部分的类成员变量应该定义为 protected 的来防止继承类使用他们。

注意，要用"int[] packets"，而不是"int packets[]"，后一种永远也不要用。

|  |
| --- |
| public void setPackets(int[] packets) { this.packets = packets; }  CounterSet(int size)  {  this.size = size;  } |

* **泛型**

代码里面的集合，List的要使用泛型List<E>来定义集合的类型，一般在实体类的多条查询或者批量操作时使用List时就可以具体定义<E>里面的参数为实体类，如List<Customer>。如果未知什么类型，可以写个E或者？号来替代。

## 编程指南

* 1. byte 数组转换到 characters

为了将 byte 数组转换到 characters，可以这么做：

"Hello world!".getBytes();

* 1. Utility 类

Utility 类（仅仅提供方法的类）应该被申明为抽象的来防止被继承或被初始化。

* 1. 初始化

如：

list = new ArrayList();

string = new String();

intNum = new Integer();

doubleNum = new Double(0.0);

floatNum = new Float(0);

date = new Date(“0000-00-00 00:00:00”);

* 1. 枚举类型

JAVA 对枚举的支持不好，但是下面的代码是一种很有用的模板：

|  |
| --- |
| class Colour {  public static final Colour BLACK = new Colour(0, 0, 0);  public static final Colour RED = new Colour(0xFF, 0, 0);  public static final Colour GREEN = new Colour(0, 0xFF, 0);  public static final Colour BLUE = new Colour(0, 0, 0xFF);  public static final Colour WHITE = new Colour(0xFF, 0xFF, 0xFF);  } |

* 1. 慎用== 和equals()方法

有时候比较两个对象之间的异同，==操作符专门用来比较两个变量的值是否相等，equals方法是用于比较两个独立对象的内容是否相同，字符串之类的选equals，数字型的选==。

* **性能** 
  1. 在写代码的时候，从头至尾都应该考虑性能问题，要注意代码的效率。
  2. 不必要的对象构造
  3. 不要在循环中构造和释放对象
  4. 使用 StringBuffer 对象

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。当我们在构造字符串的时候，我们应该用 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，那么我们应该使用 StringBuffer 对象和它的 append() 方法。如果我们用 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。

* 1. 避免太多的使用 synchronized 关键字

避免不必要的使用关键字 synchronized，应该在必要的时候再使用它，这是一个避免死锁的好方法。

* 1. 可移植性

为了保证整个系统的可移植性，尽量避免使用synchronized 关键字

* 1. PrintStream

PrintStream 已经被不赞成（deprecated）使用，用 PrintWrite 来代替它。

* 1. 避免在循环条件中使用复杂表达式

在不做编译优化的情况下，在循环中，循环条件会被反复计算，如果不使用复杂表达式，而使循环条件值不变的话，程序将会运行的更快。

9、将try/catch块移出循环

把try/catch块放入循环体内，会极大的影响性能，所以尽量放在循环体外面。

# 其他要求

* 在controller层或该层之后关闭数据库连接，不能在页面里关闭
* 在mapper的配置xml文件里面，尽量不使用过长的连接查询的sql语句，可以在resultMap里面定义多层关联的assosiation来代替关联查询。
* 数据库的表里面的字段名应尽量避免和数据库关键字相同，如change。
* 使用ajax传送数据时，注意可能会出现中文乱码的问题，应当在配置文件里面配置一个MappingJackson2HttpMessageConverter。
* 代码内部使用log4j日志器来测试代码的运行效果。
* 功能模块页面jsp统一放进内部的文件夹里面，而像404,login,register这些页面就可以统一放到WebContent文件夹下面。
* 数据的提取工作不在页面内完成
* 若是中文参数值，尽量用post方式，不用url方式
* 在将自己的代码提交到版本控制服务器之前一定要做好单元测试，使用JUnit4测试写好的测试类，不能提交含有单元测试就应该解决的bug的代码
* 异常处理要得当，不允许任何异常都是Exception，要尽量处理可以预料到的异常。
* 在使用了jdbc的事务时，处理结束后，一定要将事务提交

如：

Conn.commit();

如果中间出现了异常，要将事务回滚后再关闭连接。

Conn.close();