

**项目管理体系文件**

**Python开发编码规范**

编 撰 人：

审 核 人：

批 准 人：

批准日期：2018-1-4

保密级别：机密

文档版本：0.0.1

**中软国际信息技术有限公司**

**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。（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 的名字应该都是由一个小写单词组成。
* **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。
* **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ShipOrder.jsp。
* Java文件的文件名要与程序中的public类名相同。
* Servet文件要以Servlet做为结尾，如AddCompanyServlet.java
* 业务处理组件JavaBean要以Bean为结尾，如ProcessBean.java

## Python 文件样式

所有的 Python(\*.py) 文件都必须遵守如下的样式规则

* **头部**

开发者信息&&编码格式&&Python版本

开发者息必须在 java 文件的开头，比如：

|  |
| --- |
| # !/usr/bin/env python3 # \*coding:utf-8\*  \_\_author\_\_ = 'rockyfire' |

* Package/Imports

import导入的时候通常是单独一行。

导入应该按照以下顺序分组：  
1. 标准库导入  
2. 相关的第三方导入  
3. 特定的本地应用/库导入  
在每个导入组之间放一行空行。

推荐绝对导入，但是绝对导入过于冗长，使用明确的相对导入

|  |
| --- |
| package hotlava.net.stats;  import os  import django-rst  from .model import Goods |

* **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、。

|  |
| --- |
| *"""*  *DestroyModelMixin*  *Destroy Model Mixin*  *"""* |

接下来是类定义，Python的类只有继承没有实现接口

|  |
| --- |
| class CategoryViewSet(mixins.ListModelMixin, mixins.RetrieveModelMixin, viewsets.GenericViewSet): |

* **Class Fields**

接下来是类的成员变量：

|  |
| --- |
| queryset = GoodsCategory.objects.filter() |

* **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。

|  |
| --- |
| /\*\*  \* Get the counters  \* @return an array containing the statistical data. This array has been  \* freshly allocated and can be modified by the caller.  \*/  public int[] getPackets() { return copyArray(packets, offset); }  public int[] getBytes() { return copyArray(bytes, offset); }  public int[] getPackets() { return packets; }  public void setPackets(int[] packets) { this.packets = packets; } |

其它的方法不要写在一行上

* **构造函数**

接下来是构造函数，它应该用递增的方式写（比如：参数多的写在后面）。 访问类型 ("public", "private" 等.) 和 任何 "static", "final" 或 "synchronized" 应该在一行中。

|  |
| --- |
| Public CounterSet (int size){  this.size = size;  } |

* **克隆方法**

如果这个类是可以被克隆的，那么下一步就是 clone 方法：

|  |
| --- |
| public Object clone() {  try {  CounterSet obj = (CounterSet)super.clone();  obj.packets = (int[])packets.clone();  obj.size = size;  return obj;  }catch(CloneNotSupportedException e) {  throw new InternalError("Unexpected CloneNotSUpportedException: " + e.getMessage());  }  } |

* **类方法**

下面开始写类的方法：

|  |
| --- |
| /\*\*  \* Set the packet counters  \* (such as when restoring from a database)  \*/  protected final  void setArray(int[] r1, int[] r2, int[] r3, int[] r4)  throws IllegalArgumentException  {  //  // Ensure the arrays are of equal size  //  if (r1.length != r2.length || r1.length != r3.length || r1.length != r4.length)  throw new IllegalArgumentException("Arrays must be of the same size");  System.arraycopy(r1, 0, r3, 0, r1.length);  System.arraycopy(r2, 0, r4, 0, r1.length);  } |

* toString 方法

无论如何，每一个类都应该定义 toString 方法：

|  |
| --- |
| public  String toString() {  String retval = "CounterSet: ";  for (int i = 0; i < data.length(); i++) {  retval += data.bytes.toString();  retval += data.packets.toString();  }  return retval;  }  } |

* 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 unix 的格式，而不是 windows 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**

缩进应该是每行4个空格. 不要在源文件中保存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.

如果你使用 UltraEdit 作为 Java 源代码编辑器的话，你可以通过如下操作来禁止保存Tab字符, 方法是通过 UltrEdit中先设定 Tab 使用的长度是4个空格，然后用 Format|Tabs to Spaces 菜单将 Tab 转换为空格。

* **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2个字符.

* 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. 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

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），不允许用import xxx.\*;格式。

* **变量**

定义及方法的返回值不允许使用全包名，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，只能用public或(default),而且一个文件中最多只能有一个public类。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。

* **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。必须象下面一样写：

|  |
| --- |
| FileOutputStream fos = new FileOutputStream(projectFile);  project.save(fos, "IDE Project File");  fos.close(); |

* **Clone**

下面是一种有用的方法：

|  |
| --- |
| implements Cloneable  public  Object clone()  {  try {  ThisClass obj = (ThisClass)super.clone();  obj.field1 = (int[])field1.clone();  obj.field2 = field2;  return obj;  } catch(CloneNotSupportedException e) {  throw new InternalError("Unexpected CloneNotSUpportedException: " + e.getMessage());  }  } |

* **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，而不要将她定义为 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她。

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;  } |

## 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

下面的代码是一种很好的初始化数组的方法：

objectArguments = new Object[] { arguments };

* 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);  } |

这种技术实现了RED, GREEN, BLUE 等可以象其他语言的枚举类型一样使用的常量。 他们可以用 '==' 操作符来比较。 但是这样使用有一个缺陷：如果一个用户用这样的方法来创建颜色 BLACK

new Colour(0,0,0)

那么这就是另外一个对象，'=='操作符就会产生错误。她的 equal() 方法仍然有效。由于这个原因，这个技术的缺陷最好注明在文档中，或者只在自己的包中使用。

* **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 对象和她的 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她，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 来代替她。

# 其他要求

* 在controll层或该层之后关闭数据库连接，不能在页面里关闭
* 不能用select \* 的查询语句
* 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，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。