**简介(Introduction)**

Azolla编程规范是Azolla.org团队开发所参考的标准之一，其目的是使得团队开发人员编写代码的规范化。

**1．为什么要有编程规范**

编程规范对于一个编码人员乃至整个开发团队而言都显得尤为重要，其中原因有：

1.在一个软件的生命周期中，有80%的花费用于维护。

2.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。

3.编程规范可以提高软件代码的可读性，可以让程序员尽快而彻底地理解软件中的代码。

为了执行规范，每个Azolla.org团队成员必须学习并遵守。

**2．版权声明**

本文档所有权归Azolla.org所有。

主要贡献者：ShaneKing

本文档现由ShaneKing进行维护。

有关评论意见及版权问题请致邮：support@azolla.org

**第一章 命名规范**

**1．包命名(Packages)**

一个唯一包名的前缀总是全部小写的ASCII字母并且是一个顶级域名，通常是com，edu，gov，mil，net，org，或1981年ISO 3166标准所指定的标识国家的英文双字符代码。包名的后续部分根据不同机构各自内部的命名规范而不尽相同。这类命名规范可能以特定目录名的组成来区分部门(department)，项目(project)，机器(machine)，或注册名(login names)。

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。对于全局包，将你的 Internet 域名反转并接上包。

Azolla包命名规定：org.azolla.项目名.模块名

例如：org.azolla.project.module

另外，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。

**2．类命名(Classes)**

类名是个一**名词**，采用大小写混合的方式，每个单词的首字母大写。尽量使你的类名简洁而富于描述。使用完整单词，避免缩写词(除非该缩写词被更广泛使用，像URL，HTML)

Azolla类命名规定：首字母大写的名词，名词词组

例如：Hello，HelloWorld

**3．接口命名(Interfaces)**

接口名为**形容词**，采用完整的英文描述符说明接口封装，所有单词的第一个字母大写。

Azolla接口命名规定：首字母大写的形容词，形容词词组

例如：Closeable，Wrapper

**4．方法命名(Methods)**

方法名是一个**动名词**，采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。

**4.1获取方法命名**

类的获取方法（一般具有返回值）一般要求被方法名使用被访问字段名，前面加上前缀get。

例如：getFirstName()，getOneMenuById()

**4.2判断方法命名**

类的布尔型的判断方法一般要求方法名使用单词 is 做前缀。

例如： isPersistent()，isString()，isUser()

或者使用具有逻辑意义的单词。

例如：equal 或 equals

**4.3设置方法命名**

类的设置方法（一般返回类型为 void）。被访问字段名的前面加上前缀 set。

例如：setFirstName()，setWarpSpeed()

**4.4普通方法命名**

类的普通方法一般采用完整的英文描述说明成员方法功能， 第一个单词尽可能采用一个生动的动名词，第一个字母小写。

例如： openFile()，addAccount()

**4.5构造方法命名**

构造方法应该用递增的方式写（比如：参数多的写在后面）。

例如：

public CounterSet(){}

public CounterSet(int size){ this.size = size;}

**4.6 toString方法**

一般情况下，每一个类都应该定义 toString 方法。

例如：public String toString() {…}

**4.7 main方法**

一般应考虑置入一个main()方法，其中包含用于测试那个类的代码，如果包含了main() 方法, 那么它应该写在类的底部。

**5．变量命名(Variables)**

在本规则中，变量表示一个类属性（attribute/property）或一个类方法中的变量。变量可以是简单数据类型，如整数或浮点数，也可以是一个对象如客户帐户、操作员等。

本变量命名规则还包括一类特殊的变量：枚数数据常量。

**5.1简单数据类型命名**

循环计数器在不影响程序可读性的前提下，可以使用传统的变量命名方式，如：i、m、n等。

除计数器以外的简单数据类型的变量由小写字母前缀+大写字母起头的英文单词（或单词缩写）组成。当然，循环计数器也可以使用这种命名方式。

例如：nLoopCounter

数组在变量名在最后加’Array’。

**5.2类实例变量命名**

1．对于可以使用单个单词表示并且该单词与类名一致（不含类名的前缀）的类实例，可以使用全部是小写字母的实例名。

Azolla类实例变量命名规定：类名

例如：hello，helloword

2．其它情况下使用以小写字母表示的类名（也可以是类名的缩写）和以大写字母起始的名称组成的标识名。

例如：hellowordByType，hellowordByUser

3．类实例数组或集合的命名

可以在前两种情况的基础上用单词的类型形式表示。

例如：hellowords，hellowordArray，hellowordList

鼓励使用数字。

例如：helloword4User，helloword2User

**5.3枚举数据常量命名**

枚举数据常量全大写。

**5.4常量命名**

类常量和ANSI常量的声明，应该全部大写，单词间用下划线隔开 (尽量避免ANSI常量，容易引起错误)。

例如：

static final int MIN\_WIDTH = 4;

static final int MAX\_WIDTH = 999;

static final int GET\_THE\_CPU = 1;

**第二章 注释规范**

**（修改代码时请同时修改注释，不然请删除注释）**

哪些地方需要注释：

类的目的（即类所完成的功能）、设置接口的目的以及应如何被使用、成员方法注释（对于设置与获取成员方法，在成员变量已有说明的情况下，可以不加注释）；

普通成员方法要求说明完成什么功能，参数含义是什么？返回什么？

普通成员方法内部注释（控制结构、代码做了些什么以及为什么这样做，处理顺序等）、实参和形参的含义以及其他任何约束或前提条件、字段或属性描述。

而对于局部变量，如无特别意义的情况下不加注释。

**1．文件注释**

遵循JavaDoc的规范，声明信息等。

Azolla文件注释样式：

/\*

\* @(#)XXX.java Created at YYYY-MM-DD

\*

\* Copyright (c) YYYY-YYYY azolla.org All rights reserved.

\* Azolla PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.

\*/

例如：

/\*

\* @(#)AzollaCode.java Created at 2013-2-23

\*

\* Copyright (c) 2011-2013 azolla.org All rights reserved.

\* Azolla PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.

\*/

**2．类注释**

遵循JavaDoc的规范，在每一个源文件的开头注明该 CLASS的作用, 作简要说明, 并写上源文件的作者, 当前版本，起始版本。如果是修改别人编写的源文件，要在修改信息上注明修改者。

Azolla类注释样式：

/\*\*

\* 描述

\*

\* @author 作者邮箱

\* @since 起始版本

\*/

例如：

/\*\*

\* description

\*

\* @author sk@azolla.org

\* @since ADK1.0

\*/

**3．类方法注释**

遵循JavaDoc的规范，在每个方法的前部用块注释的方法描述此方法的作用，以及传入，传出参数的类型和作用，以及需要捕获的错误。

Azolla方法注释样式：

/\*\*

\* 描述

\*

\* @param 参数

\* @return 返回类型

\* @throws 异常类型

\*/

例如：

/\*\*

\* description

\*

\* @param args

\* @return String

\* @throws Exception

\*/

**4．行注释**

使用”//…”的注释方法来注释需要表明的内容。并且把注释的内容放在需要注释的代码的前面一行或同一行。

常用于：

1．行末注释：紧接在一个程序行最右边，对该行程序进行简要说明，注释内容在一行内写完。

例如：

程序行……; //注释内容

2．局部变量注释：原则上，一行中只允许定义一种类型的同一种用途的变量。局部变量定义必须在同一行定义语句后面加以注释，使用行末说明式的注释方法。一个变量只能用于一种用途。例如用于循环计数的变量就不要再用作记录函数的返回值。

例如：

int msgLength; //提示信息长度

3．条件判断if...else：条件分支语句末进行注释

例如：

/\*\*

\* 判断说明

\*/

if (…){ //处理说明

… …

}

else{ //处理说明

… …

}

4．条件判断switch...case...default：条件分支语句末进行注释

例如：

/\*\*

\* 判断说明

\*/

switch (…){

case …: //处理说明

… …

break;

… …

default: //处理说明

… …

}

**5．块注释**

使用”/\*\*...\*/”注释的方法来注释需要表明的内容。并且把注释的内容放在需要注释的代码的前面。

常用于：

1．详细说明式：较详细地解释后续处理的功能、算法等内容，可以有多行说明。

例如：

/\*\*

\* 说明内容

\* ……

\*/

程序行……

2．简单说明式：简要说明后续处理，注释内容在一行内写完。

例如：

/\*\* 说明内容 \*/

程序行……

3．枚举类型（常量）说明：

例如：

/\*\* 标记-格式 \*/

public static final String MESSAGE\_FORMAT = "MF";

/\*\* 标记-口令 \*/

public static final String PASSWORD = "P";

4．语句块注释：完成一种功能的一组语句组成一个语句块。语句块之间应该使用空行适当分隔。在语句块前面必须加注释，详细说明其后的语句块的功能。注释采用详细说明或简单说明的注释形式。

在语句块的注释中，必须包括详细设计文档中的所有伪码，以标明此段程序是在实现哪部分详细设计。

例如：

/\*\*

\* 功能说明一

\*/

语句

语句

……

（空行以分隔两个功能的语句块）

/\*\*

\* 功能说明二

\*/

语句

语句

……

5．条件判断if...else...：每一个条件判断语句前面使用详细说明或简单说明形式进行注释，在每一个分支内或者使用详细/简单说明形式进行注释，或者在条件分支语句后面以行末说明形式进行注释。

例如：

/\*\*

\* 判断说明

\*/

if (…){

/\*\*

\* 处理说明

\*/

… …

}

else{

/\*\*

\* 处理说明

\*/

… …

}

6．条件判断switch...case...default：与if式的判断语句类似，在每一个条件判断语句前面使用详细说明或简单说明形式进行注释，在每一个分支内或者使用详细/简单说明形式进行注释，或者在条件分支语句后面以行末说明形式进行注释。

例如：

/\*\*

\* 判断说明

\*/

switch (…){

case …:

/\*\*

\* 处理说明

\*/

… …

break;

… …

default:

/\*\*

\* 处理说明

\*/

… …

}

7．循环控制语句：循环语句块前必须使用详细说明或简单说明形式进行注释说明，在循环体内部的每一条循环控制语句或者使用详细/简单说明形式进行注释，或者使用行末说明形式进行注释说明。

例如：

/\*\*

\* 循环处理说明

\*/

for (…;…;…){

… …

break; //退出循环说明（如何种条件下退出循环）

… …

continue; //循环控制条件说明

… …

}

8．方法调用：在每一个方法调用前使用详细说明或简单说明形式进行注释，解释调用该方法的目的。

例如：

/\*\*

\* 使用得到的方法名调用合适的应用功能

\*/

output = invokeProduct(request, ac, strMethodName);

**第三章 排版缩进代码书写规范**

有一个好的程序员要有一个良好的代码书写习惯。

**1．排版缩进**

**1.1排版**

1．所有对应的大括号必须在同一竖线上且都占一行

例如：

/\*\*

\* Delete file sub files and under this directory and sub directory.

\* if this file is document or empty directory delete it.

\*

\* @param file document or directory

\* @return will return true with out false when some file delete failure

\*/

public static boolean delDirectory(File file)

{

Preconditions.checkNotNull(file);

boolean ret = true;

if(file.exists())

{

if(file.isDirectory() && file.list() != null)

{

for(File f : file.listFiles())

{

ret = ret && delDirectory(f);

}

ret = ret && file.delete();

}

else

{

ret = ret && file.delete();

}

}

return ret;

}

**1.2缩进或换行缩进**

1．大括号里的首行代码，必须在下一行，并且缩进两个空格（或一个 TAB）。

例如：

{

这里开始写代码

}

2．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：

①．在一个逗号后面断开

②．在一个操作符前面断开

③．选择较低级别(lower-level)的断开

④．新的一行应该与上一行同一级别表达式的开头处对齐

⑤．如果以上规则导致你的代码混乱或者使你的代码都堆挤在右边，那就代之以缩进8个空格。

2.1以下是两个断开算术表达式的例子。前者更好，因为断开处位于括号表达式的外边，这是个较高级别的断开。

例如1：

someMethod(longExpression1, longExpression2, longExpression3,

longExpression4, longExpression5);

var = someMethod1(longExpression1,

　 someMethod2(longExpression2,

longExpression3));

例如2：

longName1 = longName2 \* (longName3 + longName4 - longName5)

+ 4 \* longname6; //PREFFER

longName1 = longName2 \* (longName3 + longName4

- longName5) + 4 \* longname6; //不建议

2.2以下是两个缩进方法声明的例子。前者是常规情形。后者若使用常规的缩进方式将会使第二行和第三行移得很靠右，所以代之以缩进8个空格

例如3：

//CONVENTIONAL INDENTATION

someMethod(int anArg, Object anotherArg, String yetAnotherArg,

Object andStillAnother)

{

...

}

//INDENT 8 SPACES TO AVOID VERY DEEP INDENTS

private static synchronized horkingLongMethodName(int anArg,

Object anotherArg, String yetAnotherArg,

Object andStillAnother)

{

...

}

2.3 if语句的换行通常使用8个空格的规则，因为常规缩进(4个空格)会使语句体看起来比较费劲。

例如4：

//DON’T USE THIS INDENTATION

if ((condition1 && condition2)

|| (condition3 && condition4)

||!(condition5 && condition6))

{ //不好的换行

doSomethingAboutIt(); //易使这行被忽视

}

//USE THIS INDENTATION INSTEAD

if ((condition1 && condition2)

|| (condition3 && condition4)

||!(condition5 && condition6))

{

doSomethingAboutIt();

}

//OR USE THIS

if ((condition1 && condition2) || (condition3 && condition4)

||!(condition5 && condition6))

{

doSomethingAboutIt();

}

2.4有三种可行的方法用于处理三元运算表达式

例如5：

alpha = (aLongBooleanExpression) ? beta : gamma;

例如6：

alpha = (aLongBooleanExpression) ? beta

: gamma;

例如7：

alpha = (aLongBooleanExpression)

? beta

: gamma;

**2．代码书写**

1．每一行的代码不宜过长，一般以页面宽度的 80%至 90%为宜。对于连接在一起，代码较长的程序，可考虑采用分行显示的方式，第二行一般在第一行的基础上缩进两个空格（或一个TAB，这一点在书写复杂的 sql 语句时，尤其要注意！ ）。

例如：

public Vector getAgentInfo(String agent\_name, String agent\_type)

throws Exception,SQLException

2．JavaBean 中各个方法之间，一般以两行间隔，而不允许连在一起。

例如：

public void getAgent()

{

}

//第一行；

//第二行；

public int getNum()

{

}

3．定义变量时，一行只能定义一个变量。

**第四章 代码编程规范**

代码书写技巧及增强程序代码的健壮性。

**1．参数**

**1.1 null校验**

1．为防止空指针异常，在使用传入参数前应进行null校验

例如：

**public** **static** String getFileType(File file)

{

Preconditions.*checkNotNull*(file);

**return** *getFileType*(file.getName());

}

**1．异常**

**1.1 选择抛出异常则不记录日志**

错误使用如：

**catch**(Exception e)

{

*LOG*.error(e.toString(), e);

**throw** e; //既然已记录日志不应再抛出

}