

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

第零章 Azolla 特色

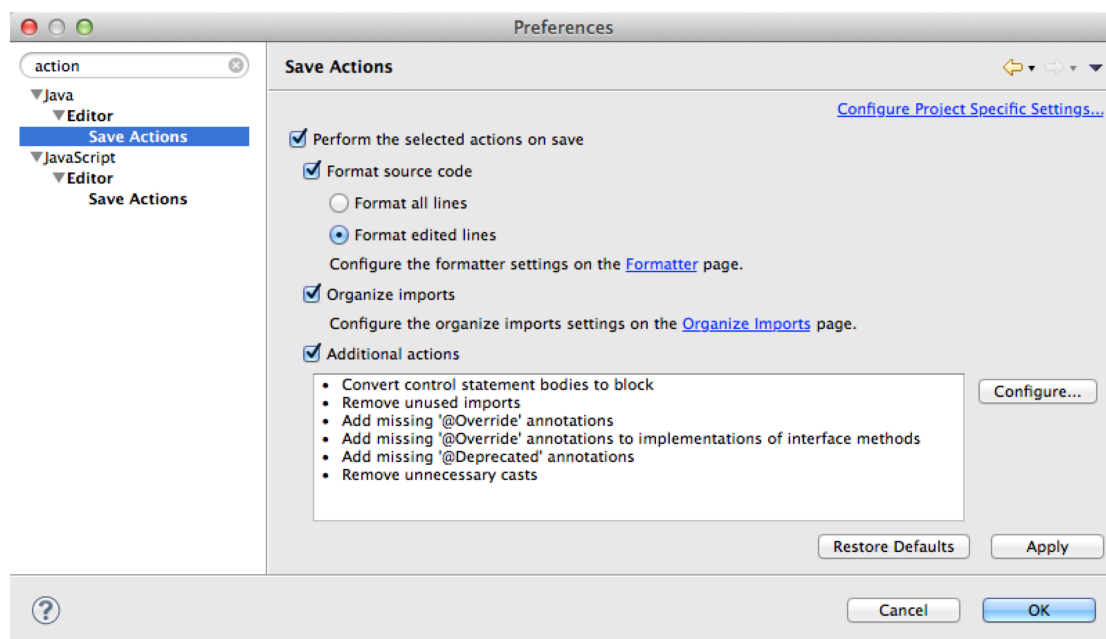
代码书写技巧及增强程序代码的健壮性。

1 . 禁用 TAB

1.1 使用四个空格代替 TAB

2 . 保存动作格式化

1.1 代码保存时同时格式化



输入文本

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**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() , getMenuById()

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` , `helloworld`

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：`helloworldByType` , `helloworldByUser`

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：`helloworlds` , `helloworldArray` , `helloworldList`

鼓励使用数字。

例如：`helloworld4User` , `helloworld2User`

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 (尽量不注释, 通过方法名反应功能)

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

有一个好的程序员要有一个良好的代码书写习惯。取：
`https://github.com/Azolla/org.azolla.open/blob/mirror/readme/dev/Azolla_Eclipse_Java_Formatter_V1.0.xml`