|  |  |
| --- | --- |
|  |  |
|  |
|  |
|  |

**数据引接标准规范说明书**

**Java程序编码规范**

北京神舟航天软件技术有限公司

（2021年10月）

修订历史记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 日期 | 版本 | 状态 | 说明 | 作者 |
| 2018年10月22日 | V0.1 | 建立 |  | 滑亚龙 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1 目的 1](#_Toc529306217)

[1.1 版权声明 1](#_Toc529306218)

[1.2 范围 1](#_Toc529306219)

[1.3 参考资料 1](#_Toc529306220)

[2 文件名 1](#_Toc529306221)

[2.1 文件后缀 1](#_Toc529306222)

[2.2 常用文件名 2](#_Toc529306223)

[3 文件组织 2](#_Toc529306224)

[3.1 文本文件编码格式 2](#_Toc529306225)

[3.2 Java源文件 2](#_Toc529306226)

[3.2.1 文件说明信息 3](#_Toc529306227)

[3.2.2 文件修订记录 3](#_Toc529306228)

[3.2.3 包和引入语句 4](#_Toc529306229)

[3.2.4 类/接口的声明 5](#_Toc529306230)

[3.2.5 文件组织 8](#_Toc529306231)

[4 代码格式 8](#_Toc529306232)

[4.1 编辑器的设置 8](#_Toc529306233)

[4.2 缩格 9](#_Toc529306234)

[4.3 行长度 9](#_Toc529306235)

[4.4 空格和空行 9](#_Toc529306236)

[4.5 换行 9](#_Toc529306237)

[4.6 语句格式 12](#_Toc529306238)

[4.6.1 简单语句 12](#_Toc529306239)

[4.6.2 复合语句 12](#_Toc529306240)

[4.6.3 返回语句 12](#_Toc529306241)

[4.6.4 if，if-else，if else-if else语句 13](#_Toc529306242)

[4.6.5 for语句 13](#_Toc529306243)

[4.6.6 while语句 14](#_Toc529306244)

[4.6.7 do-while语句 14](#_Toc529306245)

[4.6.8 switch语句 14](#_Toc529306246)

[4.6.9 try-catch语句 15](#_Toc529306247)

[4.7 声明格式 15](#_Toc529306248)

[4.7.1 每行声明变量的数量 15](#_Toc529306249)

[4.7.2 初始化 15](#_Toc529306250)

[4.7.3 布局 15](#_Toc529306251)

[4.7.4 类和接口的声明 16](#_Toc529306252)

[5 注释 17](#_Toc529306253)

[5.1 实现注释的使用 17](#_Toc529306254)

[5.1.1 块注释 17](#_Toc529306255)

[5.1.2 单行注释 18](#_Toc529306256)

[5.1.3 追踪注释 18](#_Toc529306257)

[5.1.4 行尾注释 19](#_Toc529306258)

[5.2 文档注释的使用 19](#_Toc529306259)

[5.2.1 类/接口的文档注释 21](#_Toc529306260)

[5.2.2 方法的文档注释 22](#_Toc529306261)

[5.2.3 属性的文档注释 22](#_Toc529306262)

[5.2.4 测试用例的文档注释 23](#_Toc529306263)

[5.3 实现注释和文档注释的关系 23](#_Toc529306264)

[6 命名规范 23](#_Toc529306265)

[6.1 目录、文件命名规则 23](#_Toc529306266)

[6.2 包命名规则 23](#_Toc529306267)

[6.3 三期系统命名规则 23](#_Toc529306268)

[6.4 类、接口命名规则 25](#_Toc529306269)

[6.5 成员方法命名规则 26](#_Toc529306270)

[6.6 存取方法(accessors)的命名 27](#_Toc529306271)

[6.6.1 getters方法的命名 27](#_Toc529306272)

[6.6.2 setters方法的命名 27](#_Toc529306273)

[6.7 类属性命名规则 28](#_Toc529306274)

[6.7.1 变量 28](#_Toc529306275)

[6.7.2 常量命名规则 28](#_Toc529306276)

[6.7.3 界面组件命名规则 29](#_Toc529306277)

[6.8 局部变量命名规则 29](#_Toc529306278)

[6.8.1 输入/输出流 29](#_Toc529306279)

[6.8.2 循环变量 29](#_Toc529306280)

[6.8.3 异常 30](#_Toc529306281)

[6.8.4 事件监听 30](#_Toc529306282)

[6.9 参数命名规则 30](#_Toc529306283)

[7 编程惯例 30](#_Toc529306284)

[7.1 通用惯例 30](#_Toc529306285)

[7.2 GUI编程 31](#_Toc529306286)

[7.2.1 属性窗体 31](#_Toc529306287)

[7.2.2 对齐方式： 31](#_Toc529306288)

[7.2.3 JToolBar 条 31](#_Toc529306289)

[7.2.4 SIZE 32](#_Toc529306290)

[7.2.5 字 32](#_Toc529306291)

[7.2.6 颜色(使用的是缺省值) 32](#_Toc529306292)

[7.2.7 边界 32](#_Toc529306293)

[7.2.8 组件的高度: 33](#_Toc529306294)

[7.2.9 JSplitPane 33](#_Toc529306295)

[7.2.10 树 33](#_Toc529306296)

[7.2.11 表 33](#_Toc529306297)

[7.2.12 布局 33](#_Toc529306298)

[7.2.13 建议 33](#_Toc529306299)

**Java程序编码规范（试行版）**

# 目的

对于代码，首要要求是它必须正确，能够按照程序员的真实思想去运行；第二个的要求是代码必须清晰易懂，使别的程序员能够容易理解代码所进行的实际工作。在软件工程领域，源程序的风格统一标志着可维护性、可读性，是软件项目的一个重要组成部分。而目前还没有成文的编码风格文档，以致于很多时候，程序员没有一个共同的标准可以遵守，编码风格各异，程序可维护性差、可读性也很差。通过建立代码编写规范，形成开发小组编码约定，提高程序的可靠性、可读性、可修改性、可维护性、可继承性和一致性，可以保证程序代码的质量，继承软件开发成果，充分利用资源，使开发人员之间的工作成果可以共享。

本文在参考业界已有的编码风格的基础上，描述了一个基于Java语言的项目风格，力求一种统一的编程风格，并从整体编码风格、代码文件风格、函数编写风格、变量风格、注释风格等几个方面进行阐述。

## 版权声明

本文档反映的是北京神舟航天软件技术有限公司，Java语言规范中的编码标准部分。

## 范围

本文档对如下开发内容进行了规范：

Java代码编写规范（不含jsp 规范）,金审工程三期项目所有承建单位开发的所有Java程序都应遵循此规范。

## 参考资料

[1] Oracle 公司Java 代码约定： http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

# 文件名

这部分列出了常用的文件名及其后缀。

## 文件后缀

Java程序使用下列文件后缀：

|  |  |
| --- | --- |
| **文件类别** | **文件后缀** |
| Java源文件 | .java |
| Java字节码文件 | .class |

## 常用文件名

常用的文件名包括：

|  |  |
| --- | --- |
| **文件名** | **用途** |
| xxx\_update\_list.txt | 记录软件更新的日志文件，xxx表示工具的英文名称。 |
| MANIFESZTY.MF | 记录软件的发布版本、main方法类等信息的文件。关于软件的发布版本格式请参阅《金审工程三期应用标准规范-第1部分-总体框架》。 |

# 文件组织

一个文件由被空行分割而成的段落以及标识每个段落的可选注释共同组成。超过2000行的程序难以阅读，应该尽量避免。“Java源文件范例”提供了一个布局合理的Java程序范例。

## 文本文件字符集

所有开发中使用的文本类文件字符集使用的编码为：不带BOM的UTF-8。

## Java源文件

每个Java源文件都包含一个单一的公共类或接口。若私有类和接口与一个公共类相关联，可以将它们和公共类放入同一个源文件。公共类必须是这个文件中的第一个类或接口。

Java源文件还遵循以下规则（按照从前到后的顺序）：

* 文件说明信息
* 文件修订记录
* 包名
* 引入语句
* 类或接口的说明信息（可选）
* 类或接口声明，其中又包括：
  + - 属性
    - 方法
* 内嵌类（根据需要）

### 文件说明信息

每个文件的开头都是如下文件说明：

/\*\*

\* -------------------------------------------------------------\*

\* COPYRIGHT(C) 2018 \*

\* National Audit Office of the People’s Republic Of China \*

\* \*

\* \*

\* This work contains confidential business information \*

\* and intellectual property of CNAO. \*

\* All rights reserved. \*

\* -------------------------------------------------------------\*

\*/

### 文件修订记录

文件说明信息完毕后是修改记录, 修改记录中应包括版本信息、修改日期、修改描述以及修改人，要根据修订的先后顺序从下向上排列，最后一次修订版本号不带“()”。标注如下：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Revision information:

\*

\*@version 1.02 yyyy-mm-dd description (programmer name)

\* (1.01) yyyy-mm-dd description (programmer name)

\* (1.0) yyyy-mm-dd description (programmer name)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

例如：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Revision information:

\*

\*@version 1.02 2018-10-01 redesign (xxxx)

\* (1.01) 2018-09-01 redesign, documentation (xxxx)

\* (1.0) 2018-08-22 Initial release (xxxxx)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### 包和引入语句

修改记录后则开始真正的java 程序, 首先是说明该类打包的目录，包名应按照编码规范建立或引用，不能随意编写：

* 包名全部用小写字母表示
* package 行要在 import 行之前
* 包名定义格式：gov.cnao.{项目名}.{模块名}.[子模块名]。  
  其中{}表示必须存在且只能有一个，[]表示可以有0 至多个，例如：  
  package gov.cnao.agent.requests;  
  package gov.cnao.ui.gui.oem.jobschedule;  
  package gov.cnao.ui.cmd.oem.physicalbackup;  
  等等。

import 中标准的包名要在本地的包名之前，而且按照功能用空行分隔及java 类首字母进行排序，Java 标准的包、类要放到最前面，引用的其它第三方包、类次之，引用的我们自己开发的包、类要放到最后。如果 import 行中包含了同一个包中的若干java 类，不便一一指出，则应该用 \* 来处理，但我们建议不要用\*来处理包，最好将每个包一一引入。

package hotlava.net.stats;

<空行>

import java.io.FileOutputStream;

import java.io.IOException;

import java.util.Observable;

<空行>

import javax.swing.JEditorPane;

<空行>

import org.apache.log4j.Logger;

<空行>

import com.oscar.jdbc.OscarJdbc2Connection;

......

### 类/接口的声明

import 语句后面，首先是类/接口的注释（注释的具体要求请参见“注释”部分），然后就是类/接口的声明。在类/接口中，要按照如下顺序编写代码：

|  |  |  |
| --- | --- | --- |
| **顺序号** | **类/接口声明的**  **各部分** | **注解** |
| 1 | 类/接口文档注释(/\*\*……\*/) | 该注释中所需包含的信息，参见"注释"部分 |
| 2 | 类或接口的声明 |  |
| 3 | 类/接口实现的注释(/\*……\*/)，如果有必要的话 | 该注释应包含任何有关整个类或接口的信息，而这些信息又不适合作为类/接口文档注释。 |
| 4 | 类的静态变量 | 首先是类的公共变量，随后是保护变量，再后是包一级别的变量(没有访问修饰符，access modifier)，最后是私有变量。 |
| 5 | 实例变量 | 首先是公共级别的，随后是保护级别的，再后是包一级别的(没有访问修饰符)，最后是私有级别的。 |
| 6 | 构造器 | 如果有多个构造函数，采用参数有少到多的顺序。 |
| 7 | 成员变量的存取方法 | 同一成员变量的存取方法放到一起，setter方法在前，getter方法后。 |
| 8 | clone 方法（如果需要的话） |  |
| 9 | 其它成员方法 | 这些方法应该按功能，而非作用域或访问权限，分组。例如，一个私有的类方法可以置于两个公有的实例方法之间。其目的是为了更便于阅读和理解代码。 |
| 10 | toString 方法 |  |
| 11 | equals方法 |  |
| 12 | main 方法（如果需要的话） |  |
| 13 | 内嵌类inner class |  |

例如：

/\*\*

\* description of the class

\*/

public class TheClass implements Serializable {

/\*other description of the class\*/

public static final String PUBLIC\_CONSZTYANT = “constant”;

protected static int protectedStaticVar = 0;

static int protectedStaticVar = 0;

private static int protectedStaticVar = 0;

…

<空行>

public String publicVar = “var”;

protected int protectedVar = 0;

int protectedVar = 0;

private int protectedVar = 0;

…

<空行>

<空行>

public TheClass(int parm1) {

…

}

<空行>

<空行>

public TheClass(int parm1, int param2) {

…

}

<空行>

<空行>

public TheClass (int parm1, int param2, string param3) {

…

}

<空行>

<空行>

public void setPublicVar( String publicVar) {

…

}

<空行>

<空行>

public String getPublicVar( String publicVar) {

…

}

…

<空行>

<空行>

public Object clone() {

…

}

<空行>

<空行>

otherMethod() {

…

}

…

<空行>

<空行>

public String toString() {

…

}

<空行>

<空行>

public boolean equals(Object obj) {

…

}

<空行>

<空行>

public static void main(String[] args) {

…

}

<空行>

<空行>

class innerClass {

…

}

}

### 文件组织

1. 应该用空行和注释（可选）来分割一个文件内不同的部分。
2. 文件的长度不能超过2000 行（特殊情况下，经过开发项目的项目经理批准除外）。

# 代码格式

## 编辑器的设置

开发环境使用Eclipse4.7.3a以上版本。附有Eclipse的排版格式文件，可导入使用。

## 缩格

4个空格常被作为缩进排版的一个单位。

## 行长度

尽量避免一行的长度超过100个字符，因为很多终端和工具不能很好处理。

注意：用于文档中的例子应该使用更短的行长，长度一般不超过90个字符。

## 空格和空行

适当的空格和空行能够增强代码的易读性：

* 在一行代码中，要用空格分割操作符及其两边的操作变量或常量。
* 用一个空行分割代码的不同逻辑块。
* 用两个空行分割不同的成员函数。

举例如下：

格式1（不可取）：

counter=1;

grandTotal=invoice.total()+getAmountDue();

grandTotal=Discounter.discount(grandTotal,this);

格式2（我们采用的，以^代表空格）：

counter^=^1;

<空行>

grandTotal^=^invoice.total()^+^getAmountDue();

grandTotal^=^Discounter.discount(grandTotal,^this);

格式1既没有在操作符的两边加空格，又没有在不同的逻辑代码块之间加空行，不可取；格式2是我们采用的格式。

## 换行

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：

* 在一个逗号后面断开
* 在一个操作符前面断开
* 宁可选择较高级别(higher-level)的断开，而非较低级别(lower-level)的断开
* 新的一行应该与上一行同一级别表达式的开头处对齐
* 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。

以下是断开方法调用的一些例子：

someMethod(longExpression1, longExpression2, longExpression3,

longExpression4, longExpression5);

var = someMethod1(longExpression1,

someMethod2(longExpression2,

longExpression3));

/\*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。\*/

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

+ 4 \* longname6; //PREFFER

longName1 = longName2 \* (longName3 + longName4

- longName5) + 4 \* longname6; //AVOID

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

//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) {

...

}

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。比如：

//DON’T USE THIS INDENTATION

if ((condition1 && condition2)

|| (condition3 && condition4)

||!(condition5 && condition6)) { //BAD WRAPS

doSomethingAboutIt(); //MAKE THIS LINE EASY TO MISS

}

//USE THIS INDENTATION INSZTYEAD

if ((condition1 && condition2)

|| (condition3 && condition4)

||!(condition5 && condition6)) {

doSomethingAboutIt();

}

//OR USE THIS

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

||!(condition5 && condition6)) {

doSomethingAboutIt();

}

这里有三种可行的方法用于处理三元运算表达式：

alpha = (aLongBooleanExpression) ? beta : gamma;

alpha = (aLongBooleanExpression) ? beta

: gamma;

alpha = (aLongBooleanExpression)

? beta

: gamma;

## 语句格式

方法定义的格式如下：

public TMfunction()^{

^^^^statements;

}

### 简单语句

每行至多包含一条语句，例如：

argv++; // 正确

argc--; // 正确

argv++; argc--; // 不可取!

### 复合语句

复合语句是包含在大括号中的语句序列，形如"{ 语句 }"。例如下面各段。

* 被括其中的语句应该较之复合语句缩进一个层次
* 左大括号"{"应位于复合语句起始行的行尾；右大括号"}"应另起一行并与复合语句首行对齐。
* 大括号可以被用于所有语句，包括单个语句，只要这些语句是诸如if-else或for控制结构的一部分。这样便于添加语句而无需担心由于忘了加括号而引入bug。

### 返回语句

一个带返回值的return语句不使用小括号"()"，除非它们以某种方式使返回值更为显见。例如：

return;

return myDisk.size();

return (size ? size : defaultSize);

### if，if-else，if else-if else语句

if-else语句应该具有如下格式：

if (condition) {

statements;

}

if (condition) {

statements;

} else {

statements;

}

if (condition) {

statements;

} else if (condition) {

statements;

} else{

statements;

}

注意：if语句总是用"{"和"}"括起来，避免使用如下容易引起错误的格式：

if (condition) //AVOID! THIS OMITS THE BRACES {}!

statement;

### for语句

一个for语句应该具有如下格式：

for^(initialization;^condition;^update)^{

^^^^statements;

}

当在for语句的初始化或更新子句中使用逗号时，避免因使用三个以上变量，而导致复杂度提高。若需要，可以在for循环之前(为初始化子句)或for循环末尾(为更新子句)使用单独的语句。

### while语句

一个while语句应该具有如下格式

while^(expression)^{

^^^^statements;

}

### do-while语句

一个do-while语句应该具有如下格式：

do^{

^^^^statements;

}^while^(condition);

### switch语句

一个switch语句应该具有如下格式：

switch^(case)^{

^^^^case: ^1

^^^^^^^^statemnet;

^^^^^^^^/\* falls through \*/

^^^^case: ^2

^^^^^^^^statemnet;

^^^^^^^^break;

……

^^^^default:

^^^^^^^^statement;

}

每当一个case顺着往下执行时(因为没有break语句)，通常应在break语句的位置添加注释。上面的示例代码中就包含注释/\* falls through \*/。

### try-catch语句

一个try-catch语句应该具有如下格式：

try^{

^^^^statements;

}^catch^(Exception ex)^{

^^^^statements;

}^finally^{

^^^^statements;

}

## 声明格式

### 每行声明变量的数量

一行一个声明，这样以利于写注释。如

int level; // indentation level

int size; // size of table

不要将不同类型变量的声明放在同一行，例如：

int foo, fooarray[]; //错误的声明格式!

### 初始化

尽量在声明局部变量的同时初始化。唯一不这么做的理由是变量的初始值依赖于某些先前发生的计算。

### 布局

只在代码块的开始处声明变量。（一个块是指任何被包含在大括号"{"和"}"中间的代码。）不要在首次用到该变量时才声明之。这会把注意力不集中的程序员搞糊涂，同时会妨碍代码在该作用域内的可移植性。

void myMethod() {

int int1 = 0; // beginning of method block

if (condition) {

int int2 = 0; // beginning of "if" block

...

}

}

该规则的一个例外是for循环的索引变量：

for (int i = 0; i < maxLoops; i++) { ... }

避免声明的局部变量覆盖上一级声明的变量。例如，不要在内部代码块中声明相同的变量名：

int count;

...

myMethod() {

if (condition) {

int count = 0; // 应该避免!

...

}

...

}

### 类和接口的声明

当编写类和接口是，应该遵守以下格式规则：

* 在方法名与其参数列表之前的左括号"("间不要有空格
* 左大括号"{"位于声明语句同行的末尾
* 右大括号"}"另起一行，与相应的声明语句对齐，除非是一个空语句，"}"应紧跟在"{"之后

class Sample extends Object {

int ivar1;

int ivar2;

Sample(int i, int j) {

ivar1 = i;

ivar2 = j;

}

int emptyMethod() {}

...

}

* 方法与方法之间以两个空行分隔

# 注释

注释是代码文件中非常重要的组成部分，良好的注释能够增强代码的易读性和可维护性。

在Java 代码中，有两种类型的注释：

* 实现注释（Implementation comments）：这就是一般的注释，用/\*......\*/（可以是多行）包括或者用//开始（单行注释）。
* 文档注释（Documentation comments）：这是Java 所特有的注释方式，用/\*\*......\*/包括。这种注释可以用javadoc 工具直接生成HTML 格式的说明文件。

## 实现注释的使用

Implementation comments 用于对代码逻辑进行说明，它的读者为编写、检查、维护这段代码的开发人员。对代码的注释量不应低于30%。

根据不同的注释用途，有以下三种方式。

### 块注释

块注释用于对文件、类、方法、结构、算法等进行大段的说明。

* 块注释要放在它要说明的代码的前面，缩进同它要说明的代码。
* 块注释的前面要有1到2个空行，以便区分前面的代码。
* 块注释用/\*....\*/分割。

举例如下（注意注释中的\*要对齐，\*后面注释语句前面有一个空格）：

neCount = 1;

/\*

\*^Here is a block comment.

\*/

for (i = 1; i < 100; i--) {

......

}

### 单行注释

* 单行注释用于简单说明它后面的一行代码或多行代码块，注释的内容不超过一行。
* 单行注释要放在它要说明的代码的前面，缩进同它要说明的代码。
* 单行注释的前面要有1个空行，以便区分前面的代码。
* 单行注释用/\*......\*/分割。

举例如下（注意\*后面注释语句前面有一个空格）：

if (condition) {

/\*^ Here is a Single-line comment.^\*/

.....

### 追踪注释

用于简单地注释一行代码，用/\*......\*/。

* 注释位于它所说明的代码行的后面，与代码在同一行上。
* 注释与代码之间要有足够的距离，以方便区分代码和注释。
* 如果某个代码块上有多个追踪注释，则这些注释的缩进要相同。

举例如下：

if (a == 2) {

return TRUE; /\* special case \*/

} else {

return isPrime(a); /\* works only for odd a \*/

}

## 文档注释的使用

文档注释用于对类、接口以及它们的方法、属性进行说明，它的读者为使用这些类、接口的开发人员。用javadoc 工具将这些注释生成为HTML文件后，使用它们的开发人员就可以参考HTML 文件编写自己的代码，而无需看到这些源代码。

文档注释包含在/\*\*......\*/中，格式如下：

/\*\*

\*^The Example class provides ...

\*/

public class Example {

…

}

格式要求如下：

* 文档注释要写在被说明的类、接口、方法、属性的前面。
* 文档注释要分行写，第一行的/\*\*和最后一行的\*/要分别各单独占一行。
* 第二行、第三行....的\*要与第一行的第一个\*对齐。即，后续行要比第一行再缩进1个字符。
* 注释的语句与前面的\*之间要空一个字符。
* 第一行的/的缩进要同后面代码的缩进。
* 在/\*\*......\*/中可以加入HTML 标记，以增强生成的HTML 文档的格式和易读性。

比如，

<ul>

<li>Today

<li>Tommorow

</ul>

结果为：

• Today

• Tommorow

下表是Javadoc Tag 的说明：



更详细的Javadoc的内容请参考：

<http://java.sun.com/j2se/javadoc/writingdoccomments/index.html>

### 类/接口的文档注释

* public 和protected 的类/接口必须要有Doc comments。
* 注释中首先要描述该类/接口的用途，然后可以跟着一个使用例子（可以省略），最后可以是一系列的@see 标记，指明参考的类；@version 标记表明类的版本等。

举例如下：

/\*\*

\* Class description goes here.

\* For example:(optional)

\* <pre>

\* write usage example here

\* </pre>

\*

\* @see class full name

\* @see URL

\* @version 1.2 31 2002-10-22

\*/

class TheClassName {

……

}

### 方法的文档注释

* 所有的public 和protected 的方法必须要有文档注释，包括所有的构造函数。
* 注释中首先是对该方法的说明，然后依次是参数、返回值、抛出的异常。
* 因为方法的直接参数和返回值的类型已经在方法的声明语句中写明了，所以在注释中可以不写类型。
* 如果参数或返回值有多个可以列出的取值，并且无法从参数或返回值的类型看到所有的取值，则必须用HTML 的列表标记<ul> <li>......</ul>列出所有的取值和说明。比如，某个参数的类型是我们已经定义好的枚举类，因为可以通过查看该枚举类得到取值信息，所以可以不用在此列出所有的取值。

由于类和接口之间存在implement 关系，某个类可能实现了一个或多个接口，那么接口中的方法说明可能同类中的方法说明重复。为此，我们规定如下：

* 对于一般的类和接口，由于它们之间没有固定的关系，所以，即使类和接口中的说明是重复的，也必须在类和接口的方法中都加文档注释。

### 属性的文档注释

对于public 和protected 的属性，要写文档注释。这类属性经常用于常量定义。

下面是两个例子，其中说明不多的可以用一行注释。

/\*\* classVar1 documentation comment \*/

public static int classVar1;

/\*\*

\* classVar2 documentation comment that happens to be

\* more than one line long

\*/

public static Object classVar2;

### 测试用例的文档注释

对测试用例的注释要求同一般的类、方法的注释要求。

## 实现注释和文档注释的关系

一个类、接口或方法可能既有文档注释，也有实现注释，规定如下：

* public 和protected 的类、接口和方法的总体描述、参数、返回值等都在前面以文档注释加以说明。同样的内容就不用实现注释再做重复说明了。
* 对public 和protected 的类、接口和方法，如果需要有算法、结构等方面的注释，则分别在类、接口和方法的内部写实现注释，不能写在外部。
* 对private 的类、接口和方法，在前面用块注释的方式写实现注释，对该类、接口和方法进行总体说明。

# 命名规范

## 目录、文件命名规则

* 创建的目录名称使用小写字母，所取名称尽量体现目录下文件所实现的功能。
* 文件名：类名+.java，大小写也要与类名一致。

## 包命名规则

* 包名全部是小写的ASCII字母或数字
* 包名的前缀是gov.cnao.<分系统名称>.<子系统名称>
* 工具函数类包名前缀为.util；
* 应用于表现层的类包名前缀为.ui，其中图形用户接口类包名前缀为ui.gui，命令行类包名前缀为ui.cmd，test case类包名前缀为.test。

例如：

gov.cnao.dc.util;

gov.cnao.dc,ui.gui.physicalback;

gov.cnao.dc,test.jdbc;

## 三期系统命名规则

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 项目 | 分系统 | 子系统 | 包名 | 英文全称 | 编码标识 |
| （一）国家审计大数据中心 | | | gov.cnao.bigdata | Big Data Center | JS03-A-DC |
|  |  | 数据采集子系统 | gov.cnao.bigdata.dcss | Data Collection and Submit Subsystem | JS03-A-DC-DC |
|  |  | 数据管理子系统 | gov.cnao.bigdata.dms | Data Management subsystem | JS03-A-DC-DM |
|  |  | 数据分析子系统 | gov.cnao.bigdata.das | Data Analysis subsystem | JS03-A-DC-DA |
|  |  | 数据运维子系统 | gov.cnao.bigdata.dos | Data Operational Subsystem | JS03-A-DC-DO |
|  |  | 数据服务子系统 | gov.cnao.bigdata.ds | Data Service Subsystem | JS03-A-DC-DS |
|  |  | 大数据中心数据门户 | gov.cnao.bigdata.dp | big Data center Portal Application | JS03-A-DC-DP |
|  |  | 信息资源库建设 | gov.cnao.bigdata.ir | Information resource library construction | JS03-A-DC-IR |
| （二）审计综合作业云平台 | | |  |  |  |
|  | 现场审计作业云（分）系统 |  | gov.cnao.ao | Audit Office | JS03-A-AO |
|  |  | 审计云平台子系统 | gov.cnao.ao.ac | Audit Cloud Platform | JS03-A-AO-AC |
|  |  | 审计应用运行环境子系统 | gov.cnao.ao.aa | Audit Application Runtime | JS03-A-AO-AA |
|  |  | 审计应用开发环境子系统 | gov.cnao.ao.ad | Audit Application Development | JS03-A-AO-AD |
|  |  | 审计应用仓库子系统 | gov.cnao.ao.aw | Audit Application Warehouse | JS03-A-AO-AW |
|  |  | 审计作业应用集成 | gov.cnao.ao.ai | Audit Application Integration | JS03-A-AO-AI |
|  |  | 综合门户子系统 | gov.cnao.ao.cp | Audit Office-Comprehensive portal | JS03-A-AO-CP |
|  | 联网审计（分）系统 |  | gov.cnao.ac | Audit Continue | JS03-A-AC |
| （三）国家审计数字化管理平台 | 国家审计综合平台 |  | gov.cnao.cc | Control Center | JS03-A-CC |
| 审计管理分系统 |  | gov.cnao.am | Audit Managemen | JS03-A-AM |
|  | 审计业务管理子系统 | gov.cnao.am.sm | Audit Management-Service Management | JS03-A-AM-SM |
| 行政办公管理子系统 | gov.cnao.am.oa | Audit Management-Office Automation | JS03-A-AM-OA |
| 人力资源管理子系统 | gov.cnao.am.hr | Human Resource Management Subsystem | JS03-A-AM-HR |
| 审计计划统计直报分系统 |  | gov.cnao.pr | Plan Report | JS03-A-PR |
| （四）审计综合研究系统 |  |  | gov.cnao.se | Simulation and Emulation | JS03-A-SE |
| （五）综合服务支撑系统 | 移动审计支撑分系统 |  | gov.cnao.ma | Mobile audit | JS03-A-MA |
| 审计知识管理分系统 |  | gov.cnao.km | knowledge management | JS03-A-KM |
| 标准研制和验证分系统 |  | gov.cnao.sv | Standard development and Verification | JS03-A-SV |
| 交换中心分系统 |  | gov.cnao.ec | Exchange Center | JS03-A-EC |
| 统一日志管理分系统 |  | gov.cnao.lm | Log management | JS03-A-LM |
| 统一权限管理分系统 |  | gov.cnao.pm | privilege management | JS03-A-PM |
| 业务信息和电子文件交换系统 |  | gov.cnao.fe | File Exchange | JS03-A-FE |
| OFD版式文件转换与利用服务 |  | gov.cnao.ef | Electronic File | JS03-A-EF |
| 审计专用安全浏览器 |  | gov.cnao.ab | Audit Browser | JS03-A-AB |
| 审计内网门户分系统 |  | gov.cnao.np | NEI Portals | JS03-A-NP |
| （六）运维服务支撑系统 | | | gov.cnao.om | NEI Portals | JS03-A-OM |
| （七）安全保障支撑系统 | | | gov.cnao.su | Safety Supporting | JS03-A-SU |

## 类、接口命名规则

类和接口的名字是一上名词，采用一个到多个完整的英文单词组成，单词紧靠在一起，每个单词的第一个字母大写（包括第一个单词），其余的字母小写。

对于一些业界公认的缩写（如URL，HTML，SQL等），也同样第一个字母大写，其余的字母小写，例如 SQL，虽然它本身是缩写，全部字母都应该大写，但是在类命名中，要写成 Sql。类和接口命名举例如下：

class Customer;

class OrderItem;

interface CnaoRequest

## 成员方法命名规则

方法名的第一个单词一般要用动词，即动宾语法

方法的命名采用一个到多个完整的英文单词的组合，单词紧靠在一起。第一个单词的所有字母全部小写，后面的单词的第一个字母大写，其它字母小写。

不管是否是 Bean，都要遵循 JavaBean 命名规范：getXXX(), setXXX()；

转换对象类型返回不同类型的方法：命名成 toType，比如：toString(), toArray()；

返回视图的方法：命名成 asType() 形式，比如asList()；

返回与调用此方法的对象（Wrapper Class）同值的原始类型的方法：命名成 typeValue() 形式。

如：

printDetailInfo();

save();

deleteItem(int Id);

findFactoryByName(String factorytName);

run();

runFast();

getBackground();

intValue()；

floatValue()。

常用的第一个单词如下：

* get
* set
* add
* modify
* delete
* save
* find
* is
* has
* can
* copy
* cut
* paste
* to
* as

## 存取方法(accessors)的命名

许多类都有存取类属性（fields / properties）的方法，称为存取方法。

### getters方法的命名

获取类属性的方法命名规则如下：

* 如果属性不是boolean 类型，则以get 作为开头的单词。
* 如果属性是boolean 类型，则用is 代替get 作为开头的单词。
* 对于boolean 类型，也可以用has 和can 代替is，但是后面的宾语会有所不同。参见后面的例子。

举例如下：

getFirstName();

getNeName();

isAtEnd();

isPrintable(); //用is

canPrint(); //用has

### setters方法的命名

设置属性的方法以set 开头，如果属性是集合类型，则以add 作为开头的单词。如：

setFirstName(String firstName);

setFactoryName(String factoryName);

addRole(String roleName); // 这里是向属性roles中添加一个元素，roles的类型是Vector

## 类属性命名规则

### 变量

变量命名规则类似于方法的命名规则，均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。变量名不应以下划线或美元符号开头，尽管这在语法上是允许的。

变量名应简短且富于描述。变量名的选用应该易于记忆，即能够指出其用途。尽量避免单个字符的变量名，除非是一次性的临时变量。

临时变量通常被取名为i，j，k，m和n，它们一般用于整型；c，d，e，它们一般用于字符型。

类型为集合（如ArrayList, Vector）的属性，要在名字中体现它代表的是多个值，且尽量使用泛型。

举例如下：

String firstName;

char c;

int i;

float myWidth;

List<Customer> customers; //集合类型

Vector<Equipment> equipmentItems; //集合类型

### 常量命名规则

在Java 中，常量一般作为类的static final 的属性存在，命名要求如下：

* 全部采用大写的英文单词或数字。
* 每个单词之间用下划线“\_”分割。

举例如下：

static final int DEVICE\_SIZE；

static final float MAX\_VALUE；

### 界面组件命名规则

对于界面组件(widget)的命名，要以能够表达这个组件用途的英文单词开头，后面跟着组件的类型。避免仅以数字区分（如JBuilder的默认命名法）。组件命名应注意通俗易懂，简明扼要，便于阅读，维护。

举例如下：

okButton;

fileMenuItem;

newFileMenuItem；

searchFrame //查找

sexComboBox//性别选择器

【另外说明】

JTextArea、JTextField 的命名应与其相对应的说明组件（如JLabel）一致。

JPanel 若有带标题的边界，命名应包含标题；若没有也可以以其在窗体的位置命名。

JTree 的命名若方便可包含根节点信息。

JSplitPane 的命名最好包含两侧的组件信息。

JTree 和JTable 的构造Model 与JTree 或JTable 命名一致。

## 局部变量命名规则

总体上，局部变量的命名规则同类属性的命名规则，但是，在如下四种情况下可以简化：

* 输入/输出流
* 循环变量
* 异常
* 事件监听

### 输入/输出流

* 输入流统一用变量名：in。
* 输出流统一用变量名：out。
* 输入输出流统一用变量名：inOut。

### 循环变量

循环变量用i, j, k...。

### 异常

异常在代码中经常出现，我们用字母e 来代表异常变量，存在多个异常变量时，参考如下格式：

....

} catch (IOException ioe) {

^^^^....

} catch (Exception e) {

^^^^....

}

### 事件监听

编写事件监听器类时，事件对象用evt来代表，例如：

....

new ActionListener (ButtonAction.BUTTON\_OK) {

^^^^public void actionPerformed(ActionEvent evt) {

^^^^^^^^…

^^^^}

});

## 参数命名规则

参数的命名规则完全同局部变量的命名规则。

# 编程惯例

## 通用惯例

以下是一些好的编码习惯：

1. 尽量将成员变量私有化，减少使用public 成员属性。对成员变量的存取应该用存取方法实现。
2. 禁止在main()以外的函数调用exit 函数。因为这样不给任何代码机会捕获退出，一个类似后台服务程序不应该因为一个模块的退出就退出。
3. 尽量少在代码中直接用数字。除了-1, 0, 1 等少数几个常用数字和循环变量外，尽量用使用常量来代替数字。
4. 不要在一个语句中同时给多个变量赋值，应该分成多个语句。如

str1 = str2 = “aString”; //不要这样。

应该写成：

str1 = “aString”;

str2 = “aString”;

1. 要用括号括起容易引起混淆的语句。如：

if (a == b && c == d) //AVOID

if ((a == b) && (c == d) //USE

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。
2. 不要在循环中构造和释放对象。这会造成效率降低。
3. 避免长方法。一个方法应当具有自我独立的意图，不要把几个意图放在一起，所以一个方法的代码也不能太长。

## GUI编程

### 属性窗体

属性窗体初始要全屏显示，窗体可以缩放，窗体的初始`位置this.setLocation(Toolkit.geetDefaulttoolkit().getScreen().width-this.getSize().width)/2,Toolkit.geetDefaulttoolkit().getScreen().height-this.getSi ze()height)/2);

（建议：在允许的情况下，窗体的长宽比例为0.618 黄金分割。）

### 对齐方式：

界面要求文字和组件左对齐

### JToolBar 条

JToolBar 条可设为不可移动。为了操作简便，对不熟悉电脑操作的人可以减少不必要的麻烦。

如果可移动，JToolBar 不加边界border,如果加上边界JToolBar 条上的可移动的标示就看不到了。ToolBar 的Button: Button 上图片和文字居中 ，图片的大小要相同，文字在图片正下方。

### SIZE

像JMenu.MenuItem 和Button 如无特别需要，不要设Size 值，这样它会根据其上面具体的文字的多少，自动调节其Size 的值。

### 字

整个界面上的字体统一字体 “Dialog”，字体类型 “0”（常规类型），字体大小 12，字体颜色 black（0，0，0），字体对齐方式 左对齐，数字和金额 右对齐，文字后面不加冒号。

（建议：对字体的选择以及以后的其他功能（各种事件），要专门找一台电脑只装纯Windows，装基本的JVM，因为在以前遇到只用JVM 在窗体或控件上直接书写汉字，在运行时显示方块（汉字显示不出来）。因为现在大家的机器安装的软件较多，很可能屏蔽此问题。建议测试时注意此事。）

### 颜色(使用的是缺省值)

在metal 风格下：

background 的RGB(212,208,200) ，foreground 的RGB(0,0,0)black；

获得焦点的组件显示为空心的长方形颜色的RGB(204,204,255)；

JTextComponent 类及其子类等文本组件中被选中的区域为高亮实心的长方形，

RGB(255,255,255)；

被选中Button 也为高亮显实。

### 边界

Frame 内框边界（可以在Panel 的Image 上设）值为top:5, bottom:5, left:5, right:5；

属性面板需要加标题栏Title 的，border 采用缺省titleBorder（Title 上的标题位置是线中间左对齐）；不需要加Title 的border 采用Etched (也可以根据实际情况选择合适的bordertype，但要保持整体布局的一致)。

Panel 上的组件离Panel 的边界值为top:5; bottom:0,left:5,right:5;

Panel 上的每两个组件的行间距不得小于 10 ，最大不得大于 30；

列间距的值设为 5；

（可以以这些数值为标准，不一定要盯死。可以根据实际情况设计，但不要小于和超出上面的值）

### 组件的高度:

JTextComponent 类及其子类（ JTextArea，JTextField ） 和JTextField 及其子类（JFrmattedTextField，JPasswordField）及JCombobox 的高度：25；每个组件列间距：5；

JRadioButton 和 JCheckBox 的高度为： 25；

### JSplitPane

JSplitPane 的移动条的宽度: 4 (不一定对每个左右布局的Frame 上加JSplitPane,对一些左右布局比较固定的界面可以不加)，如果需要加oneTouchExpandable()移动条的宽度为5；

### 树

树的位置在界面的左边，把树都要加到JScroPane 上，JTreeNode 最长的名和JScrollPane右边的距离不要过大。

### 表

表的每格的高度为：19，（缺省值为17，有点小，有时组件的字显示不下），每个表格的宽度可根据表格里要显示的字数和表头字数的多少来决定宽度（如果两边或上下都是表可以把表放到JScrollPane）。

### 布局

在布局上可以先用BorderLayout 把整个界面分成几个部分，根据BorderLayout 布局管理器的特点（在窗体变化时，South 和North 的宽度变化而高度不变；West 和East 的宽度不变而高度变化；而Center 时高度和宽度都变化。可以根据每个区域的特点来放合适的内容）把整Frame 分成几个Panel（根据需要在每部分中还可以使用BorderLayout 再细分），在每个Panel 上再使用XYLayout 进行详细布局，布完句后再转成GridBagLayout,然后在调节好GridBagLayout 的 Constraints 的值。

(补充：以上要求可根据实际情况而定，以最终结果为准，中间过程因人而异。)

### 建议

不要使用XYLayout 布局，这样在改变窗体时容易出错，也不好调节。

不要在一整个Frame 上这样使用GridBagLayout，这样以后修改和在添加新的组件时就要对整个Frame 重新布局。