Java 开发规范

一、引言

编码规范对于程序员而言尤为重要，有以下几个原因：

- 一个软件的生命周期中，80%的花费在于维护

- 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

- 编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码

为了规范 Java 软件开发代码及文档，方便维护，特制定 Java 软件开发规范如下。

二、文档编写要求

每个类、页面、JS 脚本、方法都必须有详细注释。要求写出模块的名称、功能、输入、输出参数介绍、创建、修改时间。

三、约定事项

为使开发者快速理解本文内容，文中有关内容以不同颜色进行标注，分别代表不同用途，相关约定如下：

红色部分表示模块或类包属于公司所有不得修改。

蓝色部分表示可以根据实际情况进行修改。

绿色部分为相应的模块名称，可根据实际情况进行变动。

粉红部分代表重要说明部分，需格外关注。

本文中所有示例项目，以“档案管理系统”项目为例，各符号代表意义如下：

公司名称：XXXXXX 公司

公司名称简拼：sdfiyon

项目名称简拼：dmis

四、文件组织(File Organization)

1 Java 源文件(Java Source Files)

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：

- 开头注释（参见“开头注释”）
- 包和引入语句（参见“包和引入语句”）
- 类和接口声明（参见“类和接口声明”）

2 开头注释(Beginning Comments)

所有的源文件都应该在开头有一个注释，其中列出类名、版本信息、日期和版权声明，其格式如下：

```
/**
```

```
 * @(#)$CurrentFile
 * 版权声明 XXXXXX 公司，版权所有 违者必究
 *
 *<br> Copyright：Copyright (c) 2010
 *<br> Company：XXXXXX 公司
 *<br> @author 文件创建者姓名（电子邮箱地址）
 *<br> 2010-07-01
 *<br> @version 1.0
*/
```

3 包和引入语句(Package and Import Statements)

在多数 Java 源文件中，第一个非注释行是包语句。在它之后可以跟引入语句。示例：

```
package java.awt;

import java.awt.peer.CanvasPeer;
```

4 类和接口声明(Class and Interface Declarations)

下表描述了类和接口声明的各个部分以及它们出现的先后次序。参见"代码范例"中一个包含注释的例子。

	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“文档注释”
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	
7	方法	这些方法应该按功能，而非作用域或访问权限，分组。

5 缩进排版(Indentation)

4 个空格常被作为缩进排版的一个单位。缩进的确切解释并未详细指定(空格 vs. 制表符)。一个制表符等于 8 个空格(而非 4 个)。

- **行长度(Line Length)**

尽量避免一行的长度超过 80 个字符，因为很多终端和工具不能很好处理之。

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 70 个字符。

- **换行(Wrapping Lines)**

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示例：

```
someMethod(longExpression1, longExpression2, longExpression3,
                    longExpression4, longExpression5);
```

以下是断开算术表达式的示例：

```
longName1 = longName2 * (longName3 + longName4 - longName5)
                      + 4 * longname6;
```

6 注释(Comments)

Java 程序有两类注释：实现注释(implementation comments)和文档注释(document comments)。

实现注释：是那些在使用/*...*/和//界定的注释。

文档注释(被称为"doc comments")：是 Java 独有的，并由/**...*/界定。文档注释可以通过 javadoc 工具转换成 HTML 文件。

实现注释的格式(Implementation Comment Formats)

程序可以有 4 种实现注释的风格：块(block)、单行(single-line)、尾端(trailing)和行末(end-of-line)。

- **块注释(Block Comments)**

块注释通常用于提供对文件，方法，数据结构和算法的描述。块注释被置于每个文件的开始处以及每个方法之前。它们也可以被用于其他地方，比如方法内部。在功能和方法内部的块注释应该和它们所描述的代码具有一样的缩进格式。

块注释之首应该有一个空行，用于把块注释和代码分割开来，比如：

```
/*
 * 这里是具体注释内容.
 */
```

- **单行注释(Single-Line Comments)**

短注释可以显示在一行内，并与其后的代码具有一样的缩进层级。如果一个注释不能在一行内写完，就该采用块注释。单行注释之前应该有一个空行。示例：

```
if (condition) {

    /* 这里是单行注释. */
    ...
}
```

- **尾端注释(Trailing Comments)**

极短的注释可以与它们所要描述的代码位于同一行，但是应该有足够的空白来分开代码和注释。若有多个短注释出现于大段代码中，它们应该具有相同的缩进。

示例：

```
if (a == 2) {
    return TRUE;              /* 尾端注释 1 */
} else {
    return isPrime(a);        /* 尾端注释 2 */
}
```

- **行末注释(End-Of-Line Comments)**

注释界定符"//"，可以注释掉整行或者一行中的一部分。它一般不用于连续多行的注释文本；然而，它可以用来注释掉连续多行的代码段。示例：

```
if (foo > 1) {

  //可以是注释掉的代码段.
  ...
}
else {
    return false;          // 这里是行末注释.
}
```

- **文档注释(Documentation Comments)**

文档注释描述 Java 的类、接口、构造器，方法，以及字段(field)。每个文档注释都会被置于注释定界符/**...*/之中，一个注释对应一个类、接口或成员。该注释应位于声明之前。

示例：

```
/**
 *<pre>类说明</pre>
 *<b>功能描述：</b>
 *    具体功能描述内容，可以为空。
 * 注意事项：
 *    具体注意事项，可以为空。
 */
public class Example { ...
```

注意：顶层的类和接口是不缩进的，而其成员是缩进的。描述类和接口的文档注释的第一行(/**)不需缩进；随后的文档注释每行都缩进 1 格(使星号纵向对齐)。成员，包括构造函数在内，其文档注释的第一行缩进 4 格，随后每行都缩进 5 格。

7 声明(Declarations)

- **每行声明变量的数量(Number Per Line)**

推荐一行一个声明，因为这样以利于写注释。示例：

```
int level;  // 变量描述 1
int size;   // 变量描述 2
```

- **类和接口的声明(Class and Interface Declarations)**

当编写类和接口时，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() {}

    ...
}
```

- 方法与方法之间以空行分隔

- **If 语句**

if 语句总是用"{"和"}"括起来，无论单行还是多行，避免使用如下容易引起错误的格式：

```
if (condition) //注意尽量避免使用此格式!
    statement;
```

应该使用的格式：

```
if (condition){ //尽量使用此格式!
    statement;
}
```

五、命名规范

1 JS/CSS 等目录命名原则

采用小驼峰式命名法，即首字母小写，其它采用驼峰式命名法；所有字母采用英文缩写时全部用小写。

JS/CSS 文件放在 WebRoot\skins\default\目录下，建立不同的分支文件夹,再依照模块分别建立目录(Folder)。

例如:

存放导入脚本或样式的文件:“WebRoot\common”;

相关 JS 脚本组件：WebRoot\component”；

相关样式表文件：WebRoot\skins\style\default\css”；

相关样式表图片：WebRoot\skins\style\default\images”。

2 JSP 目录命名原则

采用小驼峰式命名法，即首字母小写，其它采用驼峰式命名法；所有字母采用英文缩写时全部用小写。

JSP 程序都放在 WebRoot\WEB-INF\jsp\目录下,再依照模块分别建立目录(Folder)。

例如:「基础档案管理」功能的目录名称为：

“WebRoot\WEB-INF\jsp\sys\”。

3 JSP 页面命名原则

采用小驼峰式命名法，即首字母小写，其它采用驼峰式命名法。

- 查询：功能名+List.jsp
- 维护：功能名+Maintain.jsp

4 包（Package）命名原则

包名的前缀以 com 开头。包名的后续部分接下来以公司简拼.项目简拼.分层模块.功能模块名称命名。所有字母必须小写命名。

- Controller：都必须放在 com.公司简拼.项目简拼.web.ctrl 目录下，再依照模块名称分别建立 Package Name，以小写命

名。

例：「基础档案管理」功能的包名为”com.sdfiyon.emis.web.ctrl.sys”；

- Service：com. sdfiyon.emis.service.模块名称
- DAO：com. sdfiyon.emis.dao
- PO：com. sdfiyon.emis.persistent

5 类（Class）命名原则

类名是个一名词，采用大驼峰式命名法。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)。

- Controller：功能名称+Controller
- Service：功能名称+Service

6 接口（Interfaces）命名原则

接口命名跟类命名原则相同，采用大驼峰式命名法，即首字母大写，其后单词的首字母大写。

7 方法（Methods）命名原则

方法命名，采用小驼峰式命名法，即首字母小写，其后单词的首字母大写。

8 变量（Variables）命名原则

采用小驼峰式命名法，除了变量名外，所有实例，包括类，类常量，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。

9 常量（Constants）命名原则

常量的声明，应该全部大写，单词间用下划线隔开。

六、JSP 开发规范

- JSP 页面编码使用 UTF-8
- JSP 页面使用 JSTL 进行资料显示，避免使用<% %>这种方式
- 查询清单使用 ext 进行显示
- Form 表单命名原则：frm+页面名称

七、代码范例(Code Examples)

下面的例子，展示了如何合理布局一个包含单一公共类的 Java 源程序。

```
/**
 * @(#)$CurrentFile
 * 版权声明 XXXXXX 公司，版权所有 违者必究
 *
 *<br> Copyright：Copyright (c) 2010
 *<br> Company：XXXXXX 公司
 *<br> @author 作者（examples@126.com）
 *<br> 2010-07-01
 *<br> @version 1.0
*/

package java.blah;

import java.blah.blahdy.BlahBlah;

 /**
  *<pre>注释示例类</pre>
  *<b>功能描述：</b>
   *    展示类源码注释写法
   * 注意事项：
   *
   */
public class Blah extends SomeClass {
     /* 类的实现注释 */

    /** 类变量注释 */
    public static int classVar1;

     /**
       *  类变量注释内容较多时
       *  可采用此种注释
       */
    private static Object classVar2;

    /** 实例变量注释 */
    public Object instanceVar1;
```

```
    /**
      * 构造器 Blah 注释
      */
    public Blah() {
          // 实现注释
    }

    /**
      * 方法 doSomething 注释
      */
    public void doSomething() {
        // 方法实现
    }

    /**
      * 方法 doSomethingElse 注释
      * @param someParam 参数描述
      */
    public void doSomethingElse(Object someParam) {
          // 方法实现
    }
}
```

八、数据库命名规范

1 数据库(DataBase)

系统名称拼音码

2 表 (Table)

全局表：Sys_功能或用途(英文意义)

信息表：info_功能或用途(英文意义)

数据表：Data_功能或用途(英文意义)

模块表：模块名称简拼(限三位)_信息表或数据表_功能或用途(英文意义)

3 列(Columns)

列名：功能或用途(英文意义)

说明：中文名称_对应表列或取值范围_详细说明

4 视图(view)

V_功能或用途(英文意义)

5 存储过程(PROCEDURE)

SP_功能或用途(英文意义)

6 函数(Function)

F_功能或用途(英文意义)