



AIDEE®

AIDEE Java编码规范



一、JAVA 编码规范

1. 编码规范概述
2. 文件体系结构规则
3. 文件命名规则
4. 排版规则
5. 注释规则
6. 命名规则
7. 变量
8. 方法
9. 可读性
10. 程序效率
11. 质量保证

1. 编码规范概述

制定编码规范的最主要的目的是为了对产出代码的长期维护。通常负责维护代码的人大多都不是开发者本人，如果有一个统一的代码格式以及说明就可以减少混淆提高理解速度。

- 范围

编码规范定义了所有代码编写者在编写Java代码时应遵守的一些规则和习惯。

本规范采用以下术语描述：

- **规则：**编程时强制必须遵守的原则。
- **建议：**编程时必须加以考虑的原则。
- **示例：**对此规则或建议给出例子。

2. 文件体系结构规则

2.1 文件体系规则

- 规则2.1.1 JSP文件目录结构

——cgi	存放cgi程序
——cgi-bin	存放cgi编译后的程序
——css	存放样式表
——html	存放临时生成的报表文件
——images	存放图片
——include	存放系统要引用的文件
——js	存放javascript脚本
——ocx	存放ocx控件
——page	存放JSP页面
——模块分类名	
——WEB-INF	WEB 配置文件, class 文件

2. 文件体系结构规则

2.1 文件体系规则

- 建议2.1.2 Java文件目录结构

|——<模块分类目录>

|——**action**

|——**dao**

|——**form**

|——**service**

|——**vo**

模块分类

存放action（控制层）文件

存放dao（数据层）文件

存放form（展示层）文件

存放业务逻辑文件

存放数据对象文件

2. 文件体系结构规则

2.2 源文件结构规则

规则2.2.1 包和引入语句

- ✓ 先写包语句，再写引入语句。
- ✓ package行和import行之间留一行空行。
- ✓ 引入的标准包名要在本地包名之前，并且包名按照字母顺序排列。
- ✓ 如果 import 行中包含了同一个包中的类超过5个，则用 * 来处理。
- ✓ import的标准包和本地包之间留一行空行。

示例：

```
package com.sitech.crmpd.core.codegen;

import java.sql.DatabaseMetaData;
import java.sql.ResultSet;

import com.sitech.crmpd.core.config.Config;
import com.sitech.crmpd.core.jdbc.ConnectionFactory;
```

2. 文件体系结构规则

2.2 源文件结构规则

规则2.2.2 类/接口

- ✓ 先写类/接口注释，标明类/接口的用途、版本、作者等信息。
- ✓ 类/接口注释之后空一行，然后写类/接口声明。
- ✓ `extends` 和 `implements` 写在不同行。
- ✓ 异常也单写一行。

示例：

```
public class CounterSet  
    extends Observable  
    implements Cloneable  
    throws XXXXXX
```

2. 文件体系结构规则

2.2 源文件结构规则

规则2.2.3 类变量的声明顺序是 **public,protected,package,private**

- ✓ 公共变量 (Public)
- ✓ 保护变量 (Protected)
- ✓ 包一级别的变量 (没有访问修饰符)
- ✓ 私有变量 (Private)

2. 文件体系结构规则

2.2 源文件结构规则

- ✓ 规则2.2.4 变量、常量的注释应放在其上方相邻位置或右方
- ✓ 规则2.2.5 用递增的方式写构造器（比如：参数多的写在后面）
- ✓ 规则2.2.6 类变量的存取方法：get和set方法
- ✓ 规则2.2.7 如果定义main() 方法，必须将main方法写在类的底部

3. 文件命名规则

3.1 Struts配制文件命名

sc-模块名.xml

3.2 jsp文件命名

模块名_main.jsp	主文件
模块名_xxx.jsp	xxx命名详见附录A：词典规范
sxxxx	文件夹，xxxx为opcode
fxxxx_x.jsp	xxxx opcode，x序号

3.3 jsp文件命名

模块名Action.java	action文件
模块名Form.java	form文件
模块名Svc.java	service文件
表 名Dao.java	dao文件
表 名Vo.java	vo文件
表 名PK.java	主键的vo文件

4. 排版规则

4.1 方法排版规则

规则4.1.1 方法名和其后的括弧之间不应有空格

示例：

```
int getResult(int forInt, char forChar)
```

规则4.1.2 缩进采用4个空格

注意：一定要使用空格键。这样可以避免使用不同的编辑器阅读程序时，因TAB键所设置的空格数目不同而造成程序布局不整齐。

4. 排版规则

4.1 方法排版规则

规则4.1.3 在方法的局部变量声明和语句之间加一个空行
示例：

```
void method1(){  
    // 声明  
    int anInt = 0;  
    String aString = null;  
    Object anObj = null;  
  
    // 语句  
    if (condition) {  
        ...  
    }  
    ...  
}
```

4. 排版规则

4.1 方法排版规则

- ✓ 规则4.1.4 块注释或单行注释之前必须有一行空行
- ✓ 规则4.1.5 方法内的两个逻辑段之间必须有一行空行
- ✓ *建议4.1.6 在方法的每个参数之间的逗号后面加一个空格*

4. 排版规则

4.2 语句排版规则

规则4.2.1 简单语句每行至多包含一条语句

示例：

```
argv++;    // 正确  
argc--;    // 正确  
argv++; argc--;    // 不允许这样使用！！
```

规则4.2.2 复合语句被括其中的语句缩进一个层次。

4. 排版规则

4.2 语句排版规则

规则4.2.3 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。

示例：

```
for (initialization; condition; update) {  
    statements;  
}
```

4. 排版规则

4.2 语句排版规则

规则4.2.4 必须用“{”和“}”将if内的语句括起来。（即使只有一条语句的情况下）

正确：

```
if(condition){  
    System.out.println(“正确” );  
}
```

错误：

```
if(condition)  
    System.out.println(“正确” );
```


4. 排版规则

4.2 语句排版规则

规则4.2.5 在多层嵌套的if语句中，需以尾端注释的方式表示出层次关系

规则4.2.6 for语句中的表达式要用空格分开

示例：

```
for (expr1; expr2; expr3)
```

规则4.2.7 在多层嵌套的for语句中，应以尾端注释的方式表示出层次关系

4. 排版规则

4.2 语句排版规则

规则4.2.8 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。下面的示例代码中就包含注释

`/* falls through */`

示例:

一个switch语句应该具有如下格式:

```
switch (condition) {  
    case ABC:  
        statements;  
        /* falls through */  
    case DEF:  
        statements;  
        break;  
    case XYZ:  
        statements;  
        break;  
    default:  
        statements;  
        break;  
}
```

4. 排版规则

4.2 语句排版规则

- 建议4.2.9 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
- 建议4.2.10 关键词for和后面的括号之间加一个空格
- 建议4.2.11 关键词while和后面的括号之间加一个空格
- 建议4.2.12 关键词switch和后面的括号之间加一个空格

4. 排版规则

4.2 语句排版规则

建议4.2.13 关键词try catch finally和后面的括号之间加一个空格

示例：

```
try {  
    statements;  
} catch (ExceptionClass e) {  
    statements;  
}  
try {  
    statements;  
} catch (ExceptionClass e) {  
    statements;  
} finally {  
    statements;  
}
```

4. 排版规则

4.3 表达式排版规则

规则4.3.1 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

规则4.3.2 用空格分隔所有的二元运算符（除了“.”）和操作数

示例：

```
a = (a + b) / (c * d);
```

规则4.3.3 一元操作符和操作数之间不加空格

一元操作符包括负号（-）、自增（++）、自减（--）。

示例：

```
i++;  
j--;
```

4. 排版规则

规则4.3.4 强制转型后要跟一个空格

示例：

```
myMethod((byte) aNum, (Object) x);  
myMethod((int) (cp + 5), ((int) (i + 3)) + 1);
```

规则4.3.5 当三元运算符“?:”的“?”之前出现包含二元运算符的表达式时，给表达式添加一对圆括号

示例：

```
(x >= 0) ? x : -x;
```

4. 排版规则

建议4.3.6 单行不应超过80个字符

断行的原则是：

- ✓ 在逗号后。
- ✓ 在操作符前。
- ✓ 在较高的层次断行。
- ✓ 所断的行之间要对齐。
- ✓ 第一次断行比上一行缩进4个字符。

4. 排版规则

示例:

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

```
var = someMethod1(longExpression1,  
                  someMethod2(longExpression2,  
                              longExpression3));
```

```
longName1 = longName2 * (longName3 + longName4 - longName5)  
           + 4 * longname6; // 更好一些
```

```
longName1 = longName2 * (longName3 + longName4  
                        - longName5) + 4 * longname6; // 避免这样使用
```

```
alpha = (aLongBooleanExpression) ? beta : gamma;
```

```
alpha = (aLongBooleanExpression) ? beta  
       : gamma;
```

```
alpha = (aLongBooleanExpression)  
       ? beta  
       : gamma;
```


5. 注释规则

5.1 类注释规则

规则5.1.1 使用JavaDoc，列出功能、版本信息、日期、作者和版权声明

示例：

```
/**
 *
 * <p>Title: 缴费</p>
 * <p>Description: 缴费</p>
 * <p>Copyright: Copyright (c) 2006</p>
 * <p>Company: SI-TECH </p>
 * @author xxxx
 * @version 1.0
 */
```

5. 注释规则

规则5.1.2 如果对文件进行了修改，必须说明修改目的、修改日期、修改人，并变更版本信息

示例：

```
/**
 *
 * <p>Title: 缴费</p>
 * <p>Description: 缴费</p>
 * <p>Copyright: Copyright (c) 2006</p>
 * <p>Company: SI-TECH </p>
 * @author xxxx
 * @version 1.0
 * 修改日期 修改人 修改目的
 */
```

5. 注释规则

5.2 类方法注释规则

使用JavaDoc。

示例：

```
/**  
 * @param CustID: 客户ID  
 * @return 返回用户的单位信息  
 * @throws Exception  
 */
```

规则5.2.1 用中文写出每个参数和返回值的含义

规则5.2.2 当修改其他组员创建的类时，增加@author标签

5. 注释规则

5.3 块注释规则

规则5.3.1 方法内部的块注释位于所描述内容之前

规则5.3.2 块注释前留一行空行

✓ 块注释的格式如下：

示例：

```
/*  
 * 这里是块注释  
*/
```

✓ 块注释和所描述代码具有一样的缩进格式。

5. 注释规则

5.4 单行注释规则

规则5.4.1 单行注释位于所描述内容之前

规则5.4.2 单行注释之前留一行空行

✓ 单行注释的格式如下：

示例：

```
//这里是单行注释
```

规则5.4.3 单行注释和所描述代码具有一样的缩进格式

规则5.4.4 注释不能在一行写完时，就采用块注释

5. 注释规则

5.5 尾端注释规则

规则5.5.1 对变量或常量的简短注释在代码右端

规则5.5.2 代码和尾端注释之间留有足够多的空白

✓ 尾端注释的格式如下：

示例：

```
String name = null;    //这里是尾端注释
```

建议5.5.3 多个短注释出现于大段代码中时，注释要有相同的缩进

5. 注释规则

5.6 JAVA代码中需要文档化的部分

规则5.6.1 参数

- ✓ 参数类型
- ✓ 用于做什么
- ✓ 一些相关约束、限制或者是前置条件(preconditions)
- ✓ 复杂的情况下，还需包含一个示例以助说明。

5. 注释规则

规则5.6.2 变量

- ✓ 描述说明
- ✓ 复杂的情况下，需要给出一个使用例子。

示例：

```
/**  
 * The unique identifier of current record.  
 * This variable is also the primary key for the Enterprise Bean.  
 **/  
public String logID;
```


5. 注释规则

规则5.6.3 类

- ✓ 类说明（包括：作者、时间、类功能、目的等）
- ✓ 已知的bug
- ✓ 该类的开发/维护历史

示例：

```
/**  
* LogBean is an EntityBean, it represent a access record.  
* This EJBBean illustrates:  
*  
* Bean-managed persistence and Container-Managed transactions,  
* the code in this file directly accesses the data storage.  
*  
* @see Log Interface  
* @version 1.0 May. 4 2000  
* @author Steve Liu  
**/
```

5. 注释规则

规则5.6.4 接口

- ✓ 接口说明（包括：作者、时间、类功能、目的等）
- ✓ 该接口应如何使用

规则5.6.5 局部变量

- ✓ 它的用法/目的

规则5.6.6 方法内部注释

- ✓ 方法内部变量说明
- ✓ 复杂的逻辑部分的代码注释

5. 注释规则

规则5.6.7 方法

- ✓ 该方法做什么以及为什么这样做
- ✓ 该方法的传入参数说明
- ✓ 该方法的返回说明
- ✓ 已知的bugs
- ✓ 该方法所抛出的异常说明
- ✓ 该方法中代码更新的历史记录
- ✓ 该方法如何调用
- ✓ 前置条件（pre-condition）/后置条件（post-condition）说明

5. 注释规则

示例:

```
/**
 * Given a Primary Key, finds an EJBs with the Primary Key
 * (cn.com.si-tech.ejb.demo.LogPK).
 * Returns an found EJB primary keys.
 *
 * @param      pk      LogPK Primary Key class
 * @return      LogPK This is the primary key class of the entity
 *              EJB.
 * @exception   javax.ejb.FinderException If there is a query
 *              failure.
 * @exception   java.rmi.RemoteException If there is
 *              a communications or systems failure.
 * @see         LogHome
 */
public LogPK ejbFindByPrimaryKey(LogPK pk)
    throws FinderException , RemoteException
```

6. 命名规则

6.1 包命名规则

规则6.1.1 包的名字全部小写

- ✓ 基本的包按如下方式组织和命名：
 <根目录>.<一级目录>.<二级目录>.<三级目录>

示例：

```
package com.sitech.crm.common
```

- ✓ 根目录和一级目录必须使用com.sitech

6. 命名规则

6.2 类命名规则

规则6.2.1 类名用英文名称，不用汉语拼音。例如Customer，而不是KH

规则6.2.2 类名是名词，采用大小写混合的方式，每个单词的首字母大写

规则6.2.3 类名不要用复数

规则6.2.4 类名不要以“A”，“An”或“The”开头

规则6.2.5 如果不能选择适当的类名，而该类与数据库一个表紧密相关，再把数据库表名作为类名

6. 命名规则

6.2 类命名规则

规则6.2.6 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)

示例：

```
class PayCostAction;  
class BaseAction;
```

6. 命名规则

6.3 接口命名规则

规则6.3.1 接口的规则与类相同

规则6.3.2 接口名前面加“I”

示例：

```
interface IClient
```


6. 命名规则

6.4 方法命名规则

规则6.4.1 方法名是动词+名词对，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

建议6.4.2 方法名应准确描述方法的功能，不要使用无意义或含义不清的动词为方法命名

6. 命名规则

6.5 变量命名规则

规则6.5.1 变量名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

规则6.5.2 除一次性的临时变量（如for循环变量）以外，不能用单个字符的变量名

规则6.5.3 如果变量名代表容器（collection），如Array，Vector等，在变量名后加“List”

建议6.5.4 变量名要简短且富于描述，能够指出其用途

建议6.5.5 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

6. 命名规则

6.6 常量命名规则

规则6.6.1 类常量全部用大写字母，单词间用下划线隔开

示例：

```
static final int MIN_WIDTH = 4;  
static final int MAX_WIDTH = 999;
```

6. 命名规则

6.7 方法的参数命名规则

规则6.7.1 使用全英文命名。首字母小写，后续单词首字母大写

示例：

```
customer, account,  
-- or --  
aCustomer, anAccount
```

6. 命名规则

6.8 数组命名规则

规则6.7.1 将[] 放在类型后

示例：

规范： **byte[] buffer;**

非规范： **byte buffer[];**

6. 命名规则

6.9 Boolean getter 方法命名规则

规则6.9.1 所有布尔型get方法必须用单词” is”作为前缀。” is”为小写，
后
续单词首字母大写，其余字母小写
禁例：

```
isPersistent()  
isString()  
isDirty()
```

6. 命名规则

6.10 Getter 方法命名规则

规则6.10.1 方法名以“get”为前缀

示例：

```
getFirstName()  
getLastName()  
getWarpSpeed()
```

6. 命名规则

6.11 Setter 方法命名规则

规则6.11.1 方法名以“set”为前缀

示例：

```
setFirstName()  
setLastName()  
setWarpSpeed()
```


6. 命名规则

6.12 构造方法命名规则

规则6.12.1 构造方法使用类名

示例：

```
Customer()  
CustomerInfo()
```

6. 命名规则

6.13 异常命名规则

规则6.13.1 异常类名使用以“Exception”结尾的示例：

```
SQLException sqle  
ProcessingException  
MyException
```

规则6.13.2 异常实例名使用字母“e”来命名

7. 变量规范

规则7.1 不要将不同类型变量的声明放在同一行

✓ 一行声明一个变量

规则7.2 提供对实例以及类变量的get和set方法

✓ 若没有足够理由，不要把实例或类变量声明为公有。

规则7.3 不要在一个语句中给多个变量赋相同的值

示例：

```
fooBar.fChar = barFoo.lchar = 'c'; //不要这样使用!
```

7. 变量规范

规则7.4 不要将赋值运算符用在容易与相等关系运算符混淆的地方

示例：

```
if (c++ = d++) {           // 不要这样使用
    ...
}
if ((c++ = d++) != 0) {
    ...
}
```

规则7.5 局部变量不要和公共变量同名

规则7.6 声明变量的同时对变量进行初始化

8. 方法规范

规则8.1 用注释详细说明每个参数的作用、取值范围及参数间的关系

建议8.2 方法的规模限制在200行以内

建议8.3 一个方法仅完成一件功能

建议8.4 为简单功能编写方法

建议8.5 不要设计多参数方法，去掉不使用的参数

建议8.6 方法的返回值要清楚、明了，让使用者不容易忽视错误情况

建议8.7 减少方法本身或方法内的递归调用

9. 可读性

规则9.1 用括号明确表示出表达式的优先级，避免使用默认优先级

示例：

```
if (a == b && c == d)    // 避免使用!  
if ((a == b) && (c == d)) // OK
```

建议9.2 避免使用不易理解的数字，用有意义的标识来替代

- ✓ 涉及物理状态或者含有物理意义的常量，不应直接使用数字，必须用有意义的常量来代替。

建议9.3 源程序中关系较为紧密的代码应尽可能相邻

9. 可读性

建议9.4 不要使用难懂的技巧性很高的语句

示例：

```
d = (a = b + c) + r;           // 错误
```

```
a = b + c;  
d = a + r;
```

建议9.5 不要在for 循环的表达式里使用3个以上变量

✓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。

10. 程序的效率

规则10.1 不要在循环内执行重复操作

- ✓ 在循环外调用一次的，就避免在循环内进行不必要的反复调用。

示例：

```
for (int i = 0; i < dw.getRowCount(); i++) {  
    ...  
}  
应写成：  
int rows = dw.getRowCount();  
for (int i = 0; i < rows; i++) {  
    ...  
}
```

规则10.2 在对字符串有附加操作时，使用StringBuffer而非String。

- ✓ 使用StringBuffer性能会好很多

10. 程序的效率

建议10.3 显式地把已经不再被引用的对象赋为null

建议10.4 不要频繁初始化对象

- ✓ 除非必要，否则不要在循环内初始化对象。

示例：

```
for (int i = 0; i < rows; i++) {  
    CommonWrkSht commonWrkSht = new CommonWrkSht();  
    ...  
}
```

应写成：

```
CommonWrkSht commonWrkSht = new CommonWrkSht();  
for (int i = 0; i < rows; i++) {  
    ...  
}
```

11. 质量保证

11.1 质量保证规则

规则11.1 打开的数据库连接、文件在使用后必须关闭

✓ 在finally中关闭数据库连接、文件。

建议11.2 合理设计程序，避免占用太大内存

11. 质量保证

11.2 安全规则

规则11.3 不要使用不推荐的API

✓ “deprecated”标记。最好不要使用。

示例：

```
private List t_list = new List ();
```

```
t_list.addItem(str);
```

如果查一下javadoc的话，会发现建议用**add()**来代替**addItem()**

```
t_list.add(str);
```

规则11.4 不要滥用goto语句。

规则11.5 系统应具有一定的容错能力，对一些错误事件（如用户误操作等）能进行自动补救。

11. 质量保证

规则11.6

1. 常量与变量比较，常量放左边

if(1 == var) //防止写成赋值语句var = 1

if("A".equals(str)) //防止str为null, 当然, "A"应该提取为常量

2. 字符串: null, 空

- 用equals方法比较内容是否相同
- 用StringUtils.isEmpty()判断null与""
- 用StringUtils.isBlank()判断null、""、" "、" "

11. 质量保证

规则11.7

- 不要“用纸包火”
- 不能处理的异常，往外层抛
- 即便异常可能影响界面友好
 - 解决问题比界面友好更重要
- 异常有助于定位出错位置

```
try{
    //可能出异常的地方
}catch(Exception sqle){

    //也可以记录一下日志
    //转换异常，或者自定义的异常
    //总之，不能啥都不做
    throw new RuntimeException("一些说明",sqle);
}
```

```
try{
    //可能出异常的地方
}catch(Exception e){
    //不做任何处理，用纸包火
}
```

➤ 谢谢！

➤ 欢迎与我联系：

➤ 网址：www.aidee.cn

