1 Java编码基本规则

1.1 代码的组织与风格

1.1.1 代码

1.1.1.1 缩进

1) 程序块要采用缩进风格编写，缩进的空格数为 4 个，等同于一个 Tab 符，建议不使用 Tab 符。Tab 符在不同的编辑器中可能有不同的显示。

1.1.1.2 分界符(大括号‘{’和‘}’)

1) 程序的分界符开括号“{”应放置在所有者所用行的最后，其前面留一个空格
2) 闭括号“}”应独占一行并且与其所有者位于同一列
3) if, for, do, while, case, switch, default 等语句自占一行，且 if, for, do, while,switch 等语句的执行语句无论多少都要加括号{},case 的执行语句中如果定义变量必须加括号{}。

示例:

```
if (a > b) {
    doStart();
}
```

如下例子不符合规范:

```
if (a > b)
    doStart();
```

1.1.1.3 行宽

1) 较长的语句、表达式或参数（>80 字符）要分成多行书写
2) 长表达式要在低优先级操作符处划分新行，操作符放在新行之首（以便突出操作符），划分出的新行要进行适当的缩进，使排版整齐，语句可读。

示例:

```
if ((very_longer_variable1 >= very_longer_variable12)
   && (very_longer_variable3 <= very_longer_variable14)
   && (very_longer_variable5 <= very_longer_variable16)) {
   doSomething();
}
if (logger.isDebugEnabled()) {
    logger.debug("Session destroyed,call-id"
             + event.getSession().getCallId());
}
```

1.1.1.4 申明

1) 不允许把多个短语句写在一行中，即一行只写一条语句

示例：

如下例子不符合规范：

```
Object o = new Object(); Object b = null;
```

推荐一行一个声明，因为这样以利于写注释。亦即：

```
int level;  // indentation level
int size;   // size of table
```

1.1.1.5 空行

1) 相对独立的程序块之间、变量说明之后必须加空行
2) 方法之间、局部变量和它后边的语句之间、方法内的功能逻辑部分之间必须加空行

示例：

```
if (a > b) {
    doStart();
}
//此处是空行
return;
```

1.1.1.6 空格

1) 关键字之后要留空格。诸如 if、for、while 等关键字之后应留一个空格再跟左括号“（”，以突出关键字。
2) 方法名之后不要留空格，紧跟左括号“（”，以与关键字区别。
3) “，”之后要留空格，如 `Function(x, y, z);` 如果“;”不是一行的结束符号，其后要留空格，如 `for (initialization; condition; update)`。
4) 赋值操作符、比较操作符、算术操作符、逻辑操作符、位域操作符，如“=”、“+=”“>=”、“<=”、“+”、“*”、“%”、“&&”、“||”、“<<”，“^”等二元操作符的前后应当加空格。
5) 一元操作符如“!”、“~”、“++”、“--”、“&”（地址运算符）等前后不加空格。
6) 对于表达式比较长的 for 语句和 if 语句，为了紧凑起见可以适当地去掉一些空格

示例：

```
for (i=0; i<10; i++)
if ((a<=b) && (c<=d))
```

1.1.1.7 导入类

1) **为了使用其它类提供的方法，在使用之前应该将它们导入。Java 语法支持“*”通配符，但是为了避免类冲突，建议对每个使用的类都分别导入，而不使用“*”通配符。**

示例：

应该采用下面的形式声明导入类：

```
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
```

而不是采用下面形式：

```
import java.io.*;
```

为了方便，可以使用 IDE 开发工具（Eclipse 快捷方式：`Ctrl + Shift + O`）自动进行导入声明。

1.1.2 注释

1.1.2.1 规则

1) **类和接口的注释放在 `class` 或者 `interface` 关键字之前，`import` 关键字之后。注释主要是一句话功能简述与功能详细描述。类注释使用“/** */”块的注释方式。**

说明：方便 JavaDoc 收集,没有 import 可放在 package 之后。注释可根据需要列出：作者、内容、以及版本号等。功能详细描述部分说明该类或者接口的功能、作用、使用方法和注意事项，建议每次修改后增加作者和更新版本号和日期，

示例：

```
/**
 * [一句话功能简述]
 * [功能详细描述]
 * @author Administrator
 * @version [Android RCS C01, 2011-4-11]
 */
```

2) **类属性(成员变量)、公有和保护方法注释：写在类属性、公有和保护方法上面，注释方式为“/** */” 采用块注释。**

示例：

```
/**
```

```
 * 注释内容
 */
private String logType;
/**
 * 注释内容
 */
public void write()
```

3) 公有和保护方法注释内容：必须列出方法的一句话功能简述、功能详细描述、输入参数、输出参数、返回值、异常等。

格式：

```
/**
 * 〈一句话功能简述〉
 * 〈功能详细描述〉
 * @param [参数1]    [参数1说明]
 * @param [参数2]    [参数2说明]
 * @return  [返回类型说明]
 * @exception/throws [异常类型] [异常说明]
 * @see   [类、类#方法、类#成员]
 */
```

示例：

```
/**
 * [取得服务器返回的数据]
 * @param data 数据信息
 * @see com.iss.mtvclient.util.http.HttpDataHandler#setData
 */
@Override
public void setData(Object data) {
     …… code
}
```

4) 注释应与其描述的代码相近，对代码的注释应放在其上方，并与其上面的代码用空行隔开，注释与所描述内容进行同样的缩进排版。

说明：可使程序排版整齐，并方便注释的阅读与理解。

示例：

```
/*
 * 注释
 */
```

```
public void example( ) {
    // 注释
    CodeBlock One

    // 注释
    CodeBlock Two
}
```

5) 修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。

说明：我们在修改和维护他人代码的时候一定要对修改的方法加以说明，这样的话代码方便维护和管理。同时也可以让原作者知道你做了什么修改。利于代码的稳定性,同理在修改自己的方法时也要求添加必要的有用信息。例如：添加参数是作什么用的。这样方便自己后期对比此方法优势以及代码维护和评定。

6) 注释的内容要清楚、明了，含义准确，防止注释二义性。

说明：错误的注释不但无益反而有害。

7) 避免在注释中使用缩写，特别是不常用缩写。

说明：在使用缩写时或之前，应对缩写进行必要的说明。建议不采用缩写。如非是一些行业约定缩写

1.1.2.2 建议

1) 避免在一行代码或表达式的中间插入注释。

说明：除非必要，不应在代码或表达式中间插入注释，否则容易使代码可理解性变差。

2) 在代码的功能、意图层次上进行注释，提供有用、额外的信息。

说明：注释的目的是解释代码的目的、功能和采用的方法，提供代码以外的信息，帮助读者理解代码，防止没必要的重复注释信息。

示例：如下注释意义不大。

```
// 如果 receiveFlag 为真
if (receiveFlag)
```

而如下的注释则给出了额外有用的信息。

```
// 如果从连结收到消息
if (receiveFlag)
```

3) 对关键变量的定义和分支语句（条件分支、循环语句等）必须编写注释。

说明：这些语句往往是程序实现某一特定功能的关键，对于维护人员来说，良好的注释帮助更好的理解程序，有时甚至优于看设计文档。

4） 注释应考虑程序易读及外观排版的因素，使用的语言若是中、英兼有的，建议多使用中文，除非能用非常流利准确的英文表达。中文注释中需使用中文标点。方法和类描述的第一句话尽量使用简洁明了的话概括一下功能，然后加以句号。接下来的部分可以详细描述。

说明：注释语言不统一，影响程序易读性和外观排版，出于对维护人员的考虑，建议使用中文。JavaDoc 工具收集简介的时候使用选取第一句话。

5） 一些复杂的代码需要说明。

说明：出于对后期代码的维护以及对项目的理解能力，代码的可读性方面考虑。必须在复杂的代码结构上面添加有用的注释和说明。

示例：这里主要是对闰年算法的说明。

```
// 1．如果能被 4 整除，是闰年；
// 2．如果能被 100 整除，不是闰年；
// 3．如果能被 400 整除，是闰年。
```

1.2 命名

1.2.1 Java基本命名规则

1) 使用可以准确说明变量/字段/类的完整的英文描述符，不可以使用汉语拼音。

2) 采用该领域的术语。

3) 采用大小写混合，提高名字的可读性。

4) 尽量少用缩写，但如果一定要使用，就要谨慎地使用。

5) 避免使用长名字（最好不超过 15 个字母）。

6) 避免使用相似或者仅在大小写上有区别的名字。

7) 变量、方法（如果有返回值）最好加上对象类型后缀（尽量使用类型英文名称，也可以约定类型简写名称后缀）。

建议：

String 类型变量后缀：Str

int 类型变量后缀：Int

Integer 类型变量后缀：Integer

List 类型变量后缀：List

ArrayList 类型变量后缀：Alist

Float 类型变量后缀：Float

Hashtable 类型变量后缀：Htable

HashMap 类型变量后缀：Hmap

等等

1.2.2 类名和接口命名

说明：类名和接口使用类意义完整的英文描述，每个英文单词的首字母使用大写、其余字母使用小写的大小写混合法。

示例：OrderInformation, CustomerList, LogManager, LogConfig, SmpTransaction

1.2.3 成员函数命名

1) **对于存取函数中的获取函数，如果获取的字段的类型是 boolean 型，则采用 is 作为前缀，否则采用 get 作为前缀，如 getUserID，isCustomer**
2) **对于存取函数中的设置函数，统一采用 set 作为前缀，如 setUserID**

说明：成员函数的命名应采用完整的英文描述符，大小写混合使用：所有中间单词的第一个字母大写。成员函数名称的第一个单词常常采用一个有强烈动作色彩的动词。

示例：

```
private void calculateRate();
public void addNewOrder();
```

1.2.4 字段属性的命名

说明：字段是说明一个对象或者一个类的一段数据。字段可以是象字符串或者浮点数这样的基本数据类型，也可以是一个对象。命名时应采用完整的英文描述符来命名字段，对于不能用英文描述的字段可以使用汉语拼音命名。如果该字段从数据库中得来，最好直接采用数据库的字段名，但注意格式转换，如：数据库中为 `customer_name`，转化为字段属性名为 `customerName`。注：属性名使用意义完整的英文描述，第一个单词的字母使用小写，剩余单词首字母大写其余字母小写的大小写混合法。属性名不能与方法名相同。

示例：

```
private String customerName;
private int orderNumber;
private String smpSession;
```

1.2.5 命名数组和集合

同样首字母小写，以后每个单词大写。另外使用单词的复数形式。除此之外，应该根据集合的类型加上不同的后缀。下表是常用的集合命名方法：

类型	命名说明
数组[]	后缀 Array，如 namesArray
List（实现了 List 接口的类）	后缀 List，如 namesList
Set（实现了 Set 接口的类）	后缀 Set，如 namesSet

Map（实现了 Map 接口的类）	后缀 Map，如 namesMap

数组申明：

数组应该总是用下面的方式来命名：

byte[] buffer 而不是：byte buffer[]

1.2.6 命名常量

说明：常量，即不变的值，用类的静态常量字段来实现。

1) **公认的约定是，采用完整的英文大写单词，在词与词之间用下划线连接 。如：MAX_VALUE**
2) **常量名使用全大写的英文描述，英文单词之间用下划线分隔开，并且使用 static final 修饰。**

示例：

```
public static final int MAX_VALUE = 1000;
public static final String DEFAULT_START_DATE = "2001-12-08";
```

1.2.7 局部变量命名

局部变量的命名原则除了应当满足字段属性的命名原则之外，还应当满足下面两点：

3) **对于循环计数器，采用 i,j,k 命名方式**
4) **另外，使用和类变量(字段属性)不同的名称，例如，如果已定义了变量 username，则不能定义另一个名为 username 的局部变量。**

1.2.8 包的命名

包名采用全小写的形式进行命名。以 com 开头+项目名称+各模块的名称

1.2.9 建议

1) **通过对函数或过程、变量、结构等正确的命名以及合理地组织代码的结构，使代码成为自注释的。**

说明：清晰准确的函数、变量等的命名，可增加代码可读性，并减少不必要的注释。

示例：

```
getContactById()
```

2) **如果函数名超过15 个字母，可采用以去掉元音字母的方法或者以行业内约定俗成的缩写方式缩写函数名。**

示例：

```
getCustomerInformation()  改为  getCustomerInfo()
```

3) **准确地确定成员函数的存取控制符号:只是该类内部调用的函数使用 private 属**

性，继承类可以使用protected属性，同包类可以调用的使用默认属性(不加属性控制符号)，对外公开的函数使用public属性

示例：

```
protected void getUserName() {
    …..code
}

private void calculateRate() {
    ……code
}
```

4) 含有集合意义的属性命名，尽量包含其复数的意义

示例：

```
customers,  orderItem
```

1.3 表达式和语句

1.3.1 一般约定

每行应该只有一条语句

1.3.2 for语句

for 语句格式如下：

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

如果语句为空：

```
for (initialization; condition; update) ;
```

1.3.3 while语句

while 语句格式如下：

```
while (condition) {
    statements;
}
```

如果语句为空：

```
while (condition);
```

1.3.4 do-while语句

do-while 语句格式如下：

```
do {
    statements;
} while (condition);
```

1.3.5 switch语句

switch 语句，每个 switch 里都应包含 default 子语句,格式如下：

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

每当一个 case 失败时（不包括 break 语句），在 break 语句通常所在的位置增加一行注释。如以上含有 /* falls through */ 的注释的代码范例所示。

每个 switch 语句都应包含一个 default 部分。这个部分中的 break 子句虽然有点多余，但它防止了以后增加另一个 case 时会发生失败错误。

1.3.6 try-catch语句

try-catch 语句格式如下：

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

1.3.7 if-else, if-else if语句

if-else, if-else if 语句，任何情况下，都应该有“{”，“}”，格式如下：

```
if (condition) {
    statements;
}
else if (condition) {
    statements;
}
else {
    statements;
}
```