***Java*编码规范**

**一、 *Java*命名的一般性指导**

*1.*类名首字母应该大写。属性（成员变量） 、方法、对象变量以及所有标识符（如形式参

数、实际参数、局部变量）的首字母应小写，其中包含的所有单词都应紧靠在一起，而

且大写中间单词的首字母。

例如：类名： *ThisIsAClassName*属性或方法名： *thisIsMethodOrFieldName*

对象变量： *thisIsAClassVariable*

*2.  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*域名反转并接上包名，

例如： *cn.edu.usst.dingyuewei.package*

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。如果 *import*行中包含了同一个包中的不同子目录，则应

该用 *\**来处理。例如

*package hotlava.net.stats;*

*import java.io.\*;*

*import java.util.Observable;*

*import hotlava.util.Application;*

这里 *java.io.\**是用来代替 *InputStream and OutputStream*的。

*3.*接口（ *Interface* ）：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。习惯上，名字后面加上后缀 *able* ， *ible*或者 *er* 。

例如： *Contactable* ， *Prompter* 。

*4.*组件（ *Component* ）使用完整的英文描述来说明组件的用途，末端应接上组件类型。

例如： *okButton* ， *customerList* ， *fileMenu* 。

*5.*类中常用方法的命名：

*l*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* ，如 *getFirstName(), getLastName()* 。

*l*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，如 *isPersistent(),*

*isString()* 。或者使用具有逻辑意义的单词，例如 *equal*或 *equals*

*l*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* ，如

*setFirstName(),setLastName()* ， *setWarpSpeed()* 。

*l*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

一个生动的动词，第一个字母小写，如 *openFile(), addAccount()* 。

*l*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;}*

*l toString*方法：一般情况下，每一个类都应该定义 *toString*方法，其格式为 ： 

*public String toString() {…}*

*l*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。

*6.*静态常量字段（ *static  final* ）一般全部采用大写字母，单词之间用下划线分隔（也有

特例，如 *Java* 类库中关于颜色的常数没有严格地全部使用大写字母） 。如 *MIN\_BALANCE,*

*DEFAULT\_DATE* 。

*7.*循环计数变量通常采用字母 *i* ， *j* ， *k*或者 *counter* 。而数组应该总是用下面的方式来

命名： *objectType[]* 或 *byte[] buffer* 。

**二、 *Java*注释约定**

*1.*类的整体注释： 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 *,*编写日期。如果是修改别人编写的源文件，要在修改

信息上注明修改者和修改日期。

例如：

*/\*\**

*\* @* （ *#* ） *:CLASSNAME.java*

*\* @description: Description of this java*

*\* @author:  PROGRAMMER'S NAME YYYY/MM/DD*

*\* @version: Version No.*

*\* @modify:*

*\* @Copyright:*版权由 拥有

*\*/*

*2.*类中方法的注释：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。

例如：

*/\*\**

*\**方法的描述

*\**

*\**

*\*@param*参数的描述

*\*@return*返回类型的描述

*\*@exception*出错信息的描述

*\*/*

*3.*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。

*4.*块注释：使用 */\*\** 和 *\*/* 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。

*5.*注释哪些部分：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。

**三、 *Javabean*开发规范**

*1.*数据库连接规范

*l*在开发过程中，数据库连接通过调用已写好的一个数据库连接类 *JDBC*来实现。

*l*数据库的连接一般放在数据库的构造方法中建立。

*l*在每个方法中， 若对数据库操作结束， 则必须显式地调用 *JDBC* 类里的方法 *close()* ，将数据库连接关闭。

*2.*代码书写规范（一般 *Java*程序代码可参考）

有一个良好的代码书写习惯。代码编写规范的基本约定如下：

*l*每一行的代码不宜过长，一般以页面宽度的 *80%* 至 *90%* 为宜。对于连接在一起，代

码较长的程序，可考虑采用分行显示的方式，第二行一般在第一行的基础上缩进两

个空格（或一个 *TAB* ，这一点在书写复杂的 *sql*语句时，尤其要注意！ ）。

例如：

*public Vector getAgentInfo(String  agent\_name, String  agent\_type)*

*throws Exception,SQLException*

*l javabean*中各个方法之间，一般以两行间隔，而不允许连在一起。

例如：

*public void getAgent()*

*{}*

*//* 第一行；

*//* 第二行；

*public int getNum()*

*{}*

*l*大括号 *{}* 使用的规定： *{}* 在使用时，如果不是在一行代码中，则应该做到：左括号

“ *{* ”与右括号“ *}* ”上下对齐，这一点在有多个嵌套的情况下显得尤为重要。大括

号里的首行代码，必须在下一行，并且缩进两个空格（或一个 *TAB* ）。

例如：

*public void processRequest(HttpServletRequest  clientRequest){*

*String itemName, itemNum;*

*int newItemNum;*

*if(submit==null){*

*clear();*

*}else{*

*update();*

*try{*

*newItemNum = (Integer.valueOf(itemNum)).intValue();*

*}catch(Num berFormatException e){*

*itemNumber = 1;*

*}*

*}*

*}*

*l*定义变量时，同一类型的变量可以一起定义，但数量一般限定在两到三个，三个以

上则必须分开定义。变量定义与流程语句之间必须向间隔一行。

*3.*例外控制规范

在编写 *javabean*时，例外（ *exception* ）的控制一般有两种方式：

*l*一种是在方法里捕获

*l*另一种是通过 *try{}catch(Exception e){}* 的方式来捕获。

两种方法无论采用哪种均可以，但他们在使用场合还是有一些区别的。

第一种捕获方法，主要适用于对具体是哪种例外、并且在哪里会发生不太清楚的情况。

第二种捕获方法，适用于比较了解例外的发生情况。

方法返回类型规范

这里所指的方法返回类型，专指返回记录集的情况。一般在 *javabean*里返回的记录集

都是以 *ResultSet*的类型返回的，考虑到 *ResultSet*在用完以后需要关闭，如果向页面返回

*ResultSet*类型，则须在页面里关闭 *rs* ，这样会带来安全方面的隐患。为了解决这个问题，

我们提供了一个方法，将 *ResultSet* 类型转换为一个 *Vector*类型。程序员在 *javabean* 里只需

调用相应的方法，即可实现转换。

**四、 *Java*编码其它约定**

*1.  JSP* 文件命名采用完整的英文描述说明 *JSP* 所完成的功能， 尽可能包括一个生动的动词，

第一个字母小写，如： *viewMessage.jsp* 、 *editUser.jsp*或者 *forumChooser.jsp* 等。

*2.  Servlet* 类命名一般对应于所服务的对象加后缀 *Service*来命名，如： *UserService* ，

*TradeService*等。

*3.*使用 *StringBuffer*对象：在处理 *String*的时候要尽量使用 *StringBuffer*类，

*StringBuffer*类是构成 *String*类的基础。 *String*类将 *StringBuffer*类封装了起

来，（以花费更多时间为代价）为开发人员提供了一个安全的接口。当我们在构造字符

串的时候，我们应该用 *StringBuffer*来实现大部分的工作，当工作完成后将

*StringBuffer*对象再转换为需要的 *String*对象。比如：如果有一个字符串必须不断

地在其后添加许多字符来完成构造，那么我们应该使用 *StringBuffer*对象和它的

*append()*方法。如果我们用 *String*对象代替 *StringBuffer*对象的话，会花费许多

不必要的创建和释放对象的 *CPU*时间。

*4.*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它，这是一个避免死锁的好方法。 必须使用时，也尽量控制范

围，最好在块级控制。

*5.*避免使用 *java.util.Vector*类：因为 *Vector*是 *is synchronized."* ，所以使用

*java.util.Vector* 类在性能上会有所减低。

*6.*尽量使用接口而不是一个具体的类：

例如： 给定一个 *SQL* 语句， 返回一个对象的列表，实现中用 *java.util.ArrayList*实现，

于是定义方法为：

*public java.util.ArrayList getObjectItems(String sql)* ；

上面的方法存在一个问题，当 *getObjectItems* 内改用 *Vector*或 *LinkedList* 实现，外

部类必须做相应更改。一个更好的方法是定义返回值为 *java.util.AbstractList* 更合适：

*public java.util.AbstractList getObjectItems(String sql)* ；

这样即使更改实现，外部类也不必做相应更改。

*7.*避免使用索引来调用数据库中间层组件返回的结果集：

例如：

*for(int i=1;i<=dt.getRowCount();i++){*

*String field1 = dt.getField(i,0).toString();*

…

*}*

而应用字段名来存取结果集：

*for(int i=1;i<=dt.getRowCount();i++){*

*String field1 = dt.getField(i,"field1").toString();*

…

*}*

这样在数据库设计更改或查询的 *SQL*语句发生变化时，不会影响到程序的执行。另外尽

量不要使用 *\** 通配符，例如 *Select \* from tableName* ，应该具体写明字段名。