除了以下几个特例之外，命名时应始终采用完整的英文描述符。此外，一般应采用小写字母，但类名、接口名以及任何非初始单词的第一个字母要大写。   
  
1.1 一般概念   
  
　　\* 尽量使用完整的英文描述符   
  
　　\* 采用适用于相关领域的术语   
  
　　\* 采用大小写混合使名字可读   
  
　　\* 尽量少用缩写，但如果用了，要明智地使用，且在整个工程中统一   
  
　　\* 避免使用长的名字（小于 15 个字母是个好主意）   
  
　　\* 避免使用类似的名字，或者仅仅是大小写不同的名字   
  
　　\* 避免使用下划线（除静态常量等）   
  
1.2 示范   
  
　　\* 包（Package） 采用完整的英文描述符，应该都是由小写字母组成。对于全局包，将你的 Internet 域名反转并接上包名。 如：   
      java.awt   
     com.ambysoft.[www.persistence](http://www.persistence/" \t "https://www.douban.com/note/82618786/_blank)   
  
　　\* 类（Class） 采用完整的英文描述符，所有单词的第一个字母大写。 如：   
      Customer   
     SavingsAccount   
  
　　\* 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，但这不是必需的。如：   
      Contactable   
     Prompter   
  
　　\* 组件/部件（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。 如：   
      okButton   
      customerList   
      fileMenu   
  
　　\* 异常（Exception） 通常采用字母 e 表示异常。 e   
  
　　\* 类变量 字段采用完整的英文描述，第一个字母小写，任何中间单词的首字大写，如： firstName   
      lastName （驼峰式命名方式）  
  
　　\* 实参/参数 同字段/属性的命名规则   
      public void setFirstName(String firstName){   
            this.firstName = firstName;   
     }   
  
　　\* 局部变量 同字段/属性的命名规则   
  
　　 \*\* 获取成员函数 被访问字段名的前面加上前缀 get。 getFirstName(), getLastName()   
  
    \*\* 布尔型的获取成员函数 所有的布尔型获取函数必须用单词 is 做前缀。 isPersistent(), isString()   
  
    \*\* 设置成员函数 被访问字段名的前面加上前缀 set。 setFirstName(), setLastName(),setWarpSpeed()   
  
    \*\* 普通成员函数 采用完整的英文描述说明成员函数功能，第一个单词尽可能采用一个生动的动词，第一个字母小写。 openFile(), addAccount()   
  
    \* 静态常量字段（static final） 全部采用大写字母，单词之间用下划线分隔。 MIN\_BALANCE, DEFAULT\_DATE   
      
    \* 循环计数器 通常采用字母 i，j，k 或者 counter 都可以接受。 i, j, k, counter   
  
    \* 数组 数组应该总是用下面的方式来命名： byte[] buffer;   
2. Java 注释约定   
  
　　一个很好的可遵循的有关注释的经验法则是：问问你自己，你如果从未见过这段代码，要在合理的时间内有效地明白这段代码，你需要哪些信息。   
  
　　2.1. 一般概念   
  
　　\* 注释应该增加代码的清晰度   
  
　　\* 保持注释的简洁   
  
　　\* 在写代码之前写注释   
  
　　\* 注释出为什么做了一些事，而不仅仅是做了什么   
  
    2.2. 示范   
  
　　\* 文档注释   
      在紧靠接口、类、成员函数和字段声明的前面注释它们。   
     /\*\*   
    \*   
    \* 客户：客户是我们将服务和产品卖给的人或机构。   
     \*/   
  
　　\* C 语言风格   
      采用 C 语言风格的注释去掉不再使用但你仍想保留的代码。仍想保留是因为用户万一会改变想法，或者在调试过程中想让它暂时失效。如：   
    /\* 这部分代码因为已被它之前的代码取代，由 B.Gustafsson, 于 1999 年 6 月   
    \*4 日注释掉。如果两年之后还未使用，将其删除。. . .   
    \*（源代码）   
    \*/   
  
　　\* 单行   
      在成员函数内采用单行注释，来说明业务逻辑、代码段和暂时变量的声明。注释符"//"后必须紧跟一个空格，然后才是注释信息。 如：   
     // 遵照 Sarek 的规定，给所有   
     // 超过 $1000 的发货单   
     // 打 5% 的折扣。让利活   
     // 动于 1995年 2 月开始.   
  
   2.3. 注释哪些部分   
  
　　类 类的目的、即类所完成的功能，注释出采用的变量。   
  
　　接口 设置接口的目的、它应如何被使用以及如何不被使用。   
  
　　成员函数注释 对于设置与获取成员函数，在成员变量已有说明的情况下，可以不加注释；普通成员函数要求说明完成什么功能，参数含义是什么返回什么；   
  
　　普通成员函数内部注释 控制结构，代码做了些什么以及为什么这样做，处理顺序等。   
  
　　实参/参数 参数含义、及其它任何约束或前提条件   
  
　　字段/属性 字段描述   
  
　　局部变量 无特别意义的情况下不加注释   
  
3. Java 文件样式约定   
  
　　所有的 Java(\*.java) 文件都必须遵守如下的样式规则：   
  
　　1) 版权信息   
  
　　版权信息必须在 java 文件的开头，比如：   
  
　　/\*\*   
    \* Copyright @ 2000 Shanghai XXX Co. Ltd.   
    \* All right reserved.   
    \* @author: gcgmh   
    \* date: 2008-12-22   
    \*/   
  
　　其他不需要出现在 javadoc 的信息也可以包含在这里。   
  
　　2) Package/Imports   
  
　　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。   
  
　　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) Class   
  
　　接下来的是类的注释，一般是用来解释类的。   
  
　　/\*\*   
   \* A class representing a set of packet and byte counters   
   \* It is observable to allow it to be watched, but only   
   \* reports changes when the current set is complete   
   \*/   
  
　　接下来是类定义，包含了在不同的行的 extends 和 implements   
　 public class CounterSet extends Observable implements Cloneable{   
        ……   
         ……   
    }   
     
   4) Class Fields   
  
　　接下来是类的成员变量：   
  
　　/\*\*   
    \* Packet counters   
    \*/   
    protected int[] packets;   
  
　　public的成员变量必须生成文档（JavaDoc）。proceted、private和 package 定义的成员变量如果名字含义明确的话，可以没有注释。   
  
　　5) 存取方法（类的设置与获取成员函数）   
  
　　接下来是类变量的存取的方法。它只是简单的用来将类的变量赋值获取值的话，可以简单的写在一行上，如类的成员变量已经有注释，类变量的存取方法可以没有注释。   
  
　　public int[] getPackets() { return this.packets; }   
   public void setPackets(int[] packets) { this.packets = packets; }   
     ……   
  
　　要求说明的是，对于集合，加入成员函数来插入和删除项；另其它的方法不要写在一行上。   
  
　　6) 构造函数   
  
　　接下来是构造函数，它应该用递增的方式写（比如：参数多的写在后面）。   
  
　　public CounterSet(int size){ this.size = size;}   
  
　　7) 克隆方法   
  
　　如果这个类是可以被克隆的，那么下一步就是 clone 方法：   
  
　　public Object clone() { try { …… }catch(CloneNotSupportedException e) { …… }}   
  
　　8) 类方法 （类的普通成员函数）   
  
　　下面开始写类的方法：   
  
　　/\*\*   
   \* Set the packet counters   
   \* param r1 - ……   
   \* param r2 - ……   
   \* ……   
   \*/   
  protected final void setArray(int[] r1, int[] r2, int[] r3, int[] r4) throws IllegalArgumentException{   
   // Ensure the arrays are of equal size   
   ……   
  }   
  
9) toString 方法   
  
　　一般情况下，每一个类都应该定义 toString 方法：   
  
　　public String toString() { ……}   
  
　　10) main 方法   
  
　　普通类，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。   
  
　　4. Java编码其它约定   
  
　　n 文档化   
  
　　必须用 javadoc 来为类生成文档。不仅因为它是标准，这也是被各种 java 编译器都认可的方法。使用 @author 标记是不被推荐的，因为代码不应该是被个人拥有的。   
  
　　n 缩进   
  
　　缩进应该是每行2个空格。 不要在源文件中保存Tab字符， 在使用不同的源代码管理工具时Tab字符将因为用户设置的不同而扩展为不同的宽度。   
  
　　如果你使用 UltrEdit 作为你的 Java 源代码编辑器的话，你可以通过如下操作来禁止保存Tab字符， 方法是通过 UltrEdit中先设定 Tab 使用的长度室2个空格，然后用 Format|Tabs to Spaces 菜单将 Tab 转换为空格。   
  
　　n 页宽   
  
　　页宽应该设置为80字符。 源代码一般不会超过这个宽度， 并导致无法完整显示， 但这一设置也可以灵活调整。 在任何情况下， 超长的语句应该在一个逗号或者一个操作符后折行。 一条语句折行后， 应该比原来的语句再缩进2个字符。   
  
　　n {} 对   
  
　　{} 中的语句应该单独作为一行。 例如， 下面的第1行是错误的， 第2行是正确的:   
  
　　if (i>0) { i ++ }; // 错误, { 和 } 在同一行 if (i>0) { i ++ }; // 正确, 单独作为一行   
  
　　n 括号   
  
　　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。 下面的例子说明括号和空格的错误及正确使用:   
  
CallProc(AParameter); // 正确   
  
　　不要在语句中使用无意义的括号，括号只应该为达到某种目的而出现在源代码中。   
  
　　n 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。   
  
　　n 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。   
  
　　5. 一些编程建议   
  
　　n 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。   
  
　　n 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。   
  
　　n 避免使用 java.util.Vector 类   
  
　　因为"Unlike the new collection implementations, Vector is synchronized."，所以使用java.util.Vector类在性能上会有所减低。   
  
　　n 尽量使用接口而不是一个具体的类   
  
　　比方如下需求，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。   
  
　　n 避免使用索引来调用数据库中间层组件返回的结果集   
  
　　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。