### Java编码规范

**标识符命名**

注：所有标识符不能出现中文，尽量使用通俗英文，不可拼音和英文混用。

**包名**

全部使用小写字母，单词间不要用字符隔开

正例：com.axj.batchfile

反例：com.axj.batch\_file

**类名**

类命名

类名使用驼峰命名，每个自然语义的单词首字母大写，其他字母小写

正例：SendThread

反例：sendThread，Sendthread

**类后缀**

类名不同的后缀代表不同的类型：

|  |  |  |
| --- | --- | --- |
| 后缀名 | 意义 | 举例 |
| Exception | 这个类是异常类 | Axj Exception |
| Service | 表明这个类是个服务类，里面包含了给其他类提同业务服务的方法 | AccountService |
| Impl | 这个类是一个实现类，AccountServiceImpl实现AccountService | AccountServiceImpl |
| Test | 日常测试类，以测试名称开头，Test为后缀 | CodeTest |
| Enum | 这个类是枚举类，枚举成员名称需要全大写，单词间用下划线隔开 | TimeEnum |
| Inter | 这个类是一个接口 | LifeCycleInter |
| Dao | 这个类封装了数据访问方法，一般为接口 | PaymentOrderDao |
| Action | 直接处理页面请求，管理页面逻辑了类 | UpdateOrderListAction |
| Listener | 响应某种事件的类 | PaymentSuccessListener |
| Event | 这个类代表了某种事件 | PaymentSuccessEvent |
| Servlet | 一个Servlet | PaymentCallbackServlet |
| Factory | 生成某种对象工厂的类 | PaymentOrderFactory |
| Adapter | 用来连接某种以前不被支持的对象的类 | DatabaseLogAdapter |
| Job | 某种按时间运行的任务 | PaymentOrderCancelJob |
| Wrapper | 这是一个包装类，为了给某个类提供没有的能力 | SelectableOrderListWrapper |
| Bean | 这是一个POJO | MenuStateBean |
| Thread | 表示是线程类 | SendThread |
| POJO | POJO 类中布尔类型的变量，都不要加 is ，否则部分框架解析会引起序列化错误 |  |
| Abstract | 抽象类最好能用“Abstract”做为类命名的开头（前缀） | AbstractBeanFactory |

**方法名**

首字母小写，后面自然语义的单词首字母都大写

正例:closeSession

反例:CloseSession，closesession

动词在前，名词在后

正例:closeSession

反例:sessionClose

方法命名格式（方法名要准确表达业务的含义）:

|  |  |  |
| --- | --- | --- |
| 前缀名 | 意义 | 举例 |
| create | 创建 | createOrder() |
| delete | 删除 | deleteOrder() |
| add | 创建，暗示新创建的对象属于某个集合 | addPaidOrder() |
| remove | 删除 | removeOrder() |
| init或则initialize | 初始化，暗示会做些诸如获取资源等特殊动作 | initializeObjectPool |
| destroy | 销毁，暗示会做些诸如释放资源的特殊动作 | destroyObjectPool |
| open | 打开 | openConnection() |
| close | 关闭 | closeConnection()< |
| read | 读取 | readUserName() |
| write | 写入 | writeUserName() |
| get | 获得 | getName() |
| set | 设置 | setName() |
| prepare | 准备 | prepareOrderList() |
| copy | 复制 | copyCustomerList() |
| modity | 修改 | modifyActualTotalAmount() |
| calculate | 数值计算 | calculateCommission() |
| do | 执行某个过程或流程 | doOrderCancelJob() |
| dispatch | 判断程序流程转向 | dispatchUserRequest() |
| start | 开始 | startOrderProcessing() |
| stop | 结束 | stopOrderProcessing() |
| send | 发送某个消息或事件 | sendOrderPaidMessage() |
| receive | 接受消息或时间 | receiveOrderPaidMessgae() |
| respond | 响应用户动作 | responseOrderListItemClicked() |
| find | 查找对象 | findNewSupplier() |
| update | 更新对象 | updateCommission() |

**变量、参数名**

变量一定要清楚表达什么意思，一般取一些通俗有意义的名字，尽量使用简短英文单词。

命名规则:

1、首字母小写，后面自然语义的单词首字母都大写

2、只能以字母、“\_”或“$”符号作为变量名开头。

3、变量中可包含数字，但不能以数字开头。

4、除了“\_”和“$”符号以外，变量中不能包含其他特殊字符。

5、不能用class、int、String、public等java关键字做为变量名。

6、在java中变量名严格区分大小写，例如：name和Name就是两个不同的变量

7、静态变量和枚举类型字母全大写，多个单词之间用”\_”分隔，如：TIME\_EVENT

**代码注释**

**注释分类**

注：注释主要分为2种，实现注释和文档注释。

**实现注释**

实现注释使用/\*...\*/和//界定，使用/\*...\*/来进行块注释，使用//进行行注释，位于注释代码上一行。

**文档注释**

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，javadoc是j2sdk里面一个非常重要的工具，如果你按照规范在[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "https://blog.csdn.net/u013050593/article/details/_blank)的源代码里面写好注释的话，那么它就可以生成相应的文档，便于开发者察看，Myeclipse生成步骤，点击菜单栏的Project，选择generate javadoc，下一步即可。

javadoc参数定义：

@author   对类的说明 标明开发该类模块的作者

@version   对类的说明 标明该类模块的版本

@see     对类、属性、方法的说明 参考转向，也就是相关主题

@param    对方法的说明 对方法中某参数的说明

@return   对方法的说明 对方法返回值的说明

@exception  对方法的说明 对方法可能抛出的异常进行说明

3.2 注释条件（必加）

1、类的注释

2、方法的注释

3、在代码复杂处注释

4、代码修改处注释上时间和修改原因

5、全局变量注释

6、在多重嵌套和循环逻辑上注释

7、接口注释

8、调用工具类中方法的注释

**注释举例**

**类注释**

/\*\*

 \*类功能概述×××××××

 \*

 \*@author huwei

 \*@time 2017-8-20

 \*/

public class LogicManager {

......

}

**方法注释**

/\*\*

 \* 方法说明×××××××

 \*

  \* @param1 参数说明

 \* @param2 参数说明

  \* @return 返回值说明

  \* @throws Exception

  \*/

public boolean executeLogic(int param1, int param2)throws Exception{

......

return true;

}

**全局变量、字段、属性注释**

public class ReadManager {

   /\*\* 全局的计数变量\*/

   private final int count;

   /\*\* 字段/属性\*/

   private String name;

}

**代码域注释**

/\*----------开始：解析数据库数据 ------- \*/

//\1拆分数据库数据

String Row[] = sStr.split("\1");

waitRecord = new WaitRecord[Row.length];

/\* 循环存储对象到List \*/

for (int i = 0; i < Row.length; i++) {

String Col[] = Row[i].split("\2");

waitRecord[i] = new WaitRecord();

waitRecord[i].waitId = Col[0];

waitRecord[i].userId = Col[1];

waitRecord[i].username = Col[2];

waitRecord[i].longId = Col[3];

waitList.add(waitRecord[i]);

}

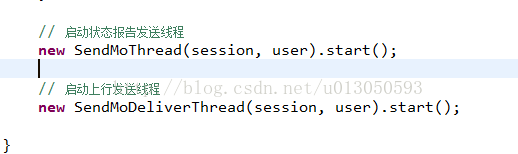
/\*----------结束：解析数据库数据 ------- \*/

**编程实践**

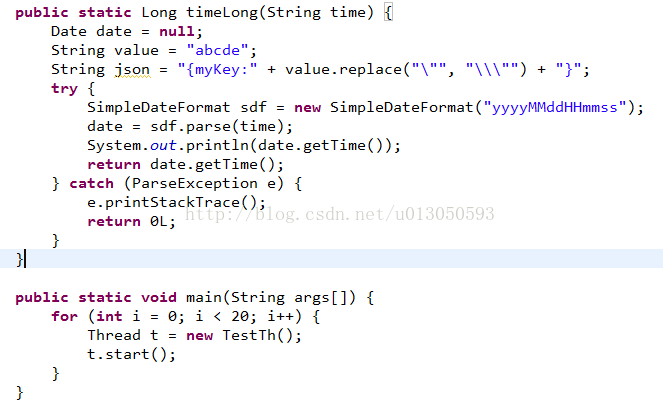
**空行**

空行一般用于分割特定区域代码，表达一段代码，总结为以下几种情况必须换行：

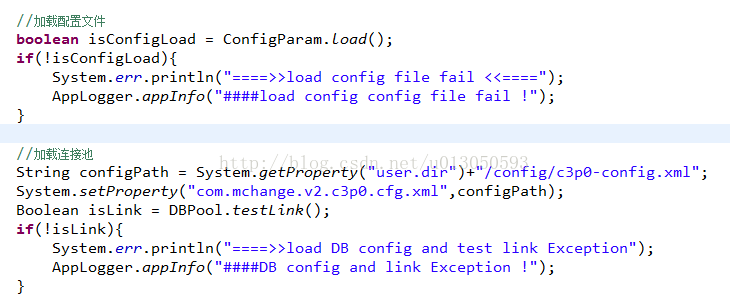
1、代码注释前得换行



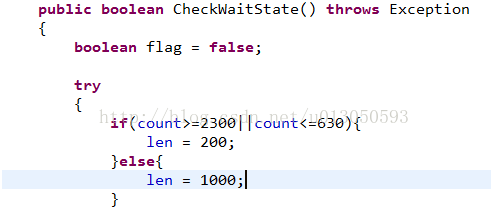
2、两个方法之间换行



3、不同逻辑段之间



4、方法里的局部变量和第一条非局部变量代码之间



**换行**

1、一行代码最大不超过120字符，超过则换行

2、大括号左右，总结如下：

（1）左大括号前不换行

（2）左大括号后换行

（3）右大括号前换行

（4）右大括号后面是else或逗号，则不换行，否则作为终止符必须换行

3、”+”号等操作符前换行，逗号后面换行，增加代码可读性

**缩进**

缩进建议采用4个空格作为一个单位，可设置一个tab键为4个空格，一般为开始一个新的块使用缩进，块结束时返回先前的缩进级别（从属关系）。

**空格**

1、运算符之间使用空格

正列：a == b ? true : false；

反例：a==b?true:false;

2、逗号后使用空格，不能在逗号前

正列：findData(a, b, c);

反例：finData(a,b ,c);

**代码习惯**

**异常抛出**

进程和线程可能出现一系列异常，异常的出现如果没有抛出可能会导致进程死掉或线程崩溃，所以需养成异常捕获抛出的习惯，避免应用因进程卡住，异常异常捕获后，需打印异常日志，否则出现异常时不好查找问题。

**减少嵌套**

代码嵌套层太多，会严重影响代码可读性，处理方法：

1、使用return跳出，减少else使用

2、条件使用逻辑运算符（&& 或 ||等）进行拼接

3、整合成子方法

**判断字符串相等**

1、java中字符串的比较是==比较引用，equals 比较值

我们经常习惯性的写上if(str1==str2)，这种写法在java中可能会带来问题

例子1：String a="abc";String b="abc"，

那么a==b将返回true。因为在java中字符串的值是不可改变的，相同的字符串在内存中只会存一份，所以a和b指向的是同一个对象；

例子2：String a=new String("abc"); String b=new String("abc");

那么a==b将返回false，此时a和b指向不同的对象。

2、用equals方法比较的是字符串的内容是否相同

example：String a=new String("abc"); String b=new String("abc"); a.equals(b);将返回true。

通常情况下，为了避免出现上述问题，判断字符串是否相等使用equals方法，具体的原因得引出常量池概念，这里不做详细描述

**字符串拼接**

拼接字符串尽量使用StringBuffer或者StringBuilder，使用”+”号每次会new一个对象，对空间和性能还是有一定影响的。

**NPE处理**

在java开发最常见的异常就是NPE（java.lang.NullPointerException）异常，需有效的规避异常，进行null判断，可能出现null模块，取集合元素，数据截取，方法返回，socket传输的数据，数据库取值等等...

**日志输出**

平时调试代码或输出日志习惯使用System.out.println()，system.out都写入到一个文件里去了，对一个文件的读写就会排队写，而且syso在当前线程，必须写完才继续下面的程序。 如果使用log工具，你的日志会发散在多个日志文件上。可以控制日志是否输出，而且log工具只是通知要写日志，而不必等日志写完。