代码规范

**1.命名方式**

使用匈牙利表示法，命名时应始终采用完整的英文描述符，一般应采用小写字母。

命名规则：

1）包的命名

包的命名一般都有小写字母组成。如 package com.util

2）类的命名

类的名字必须由大写字母开头而其他字母都小写的单词组成，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。

3）类的变量命名

变量的名字必须由一个小写字母开头，后面的单词用大写字母开头。如 userName

4）静态变量的命名

静态变量的名字应该都大写，并且指出完整含义。

5）参数的命名

参数的名字必须和变量的命名规范一致。

6）数组的命名

**2.注释规范**

1）注释要简单明了。 String username=null;// 用户

2）编写代码边注释，修改代码同时修改相应的注释，以保证注释和代码的一致性。

3）在必要的地方注释，注释量要适中。注释的内容要清楚、明了、含义准确，防止注释二义性。保持注释与其描述的代码相邻，即注释的就近原则。

4）对代码的注释应放在其上方相邻位置，不可放在下面。对数据结构的注释应放在其上方相邻位置，不可放在下面；对结构中的每个域的注释应放在此域的右方；统一结构中不同域的注释要对齐。

5）变量、常量的注释应该放在其上方相邻位置或右方。

6）全局变量要有较详细的注释，包括对其功能、功能范围、那些函数或过程存取它以及存取时注意事项等的说明。

7）在每个源文件的头部要有必要的注释信息，包括：作者；生成日期；模块功能描述；文件历史修改记录等。

/\*\*

\* 预警监控明细纳税人信息指标

\* 业务规则说明：

\* @author xx

\*/

8）在每个函数或过程的前面要有必要的注释信息。 包括：功能描述； 输入、输出及返回值说明；

**3.编程技巧**

1）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。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。

2）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。

3）避免使用java.util.Vector 类

因为此类属于单线程，所以使用它操作大量数据时，在性能上会有很大的损失。

4）面向接口编程

在程序的开发中，建议尽可能的采用面向接口进行编程，即系统业务方法都必须继承自相应的接口。系统间的方法调用也是通过接口而非具体实现类完成，期间如果业务方法发生变化或者系统应用环境发生变化，则采用新的方法实现已定义的接口，替换以前的实现即可，而无需修改已有的代码。

总之就是要对修改封闭，对扩展开放。

5）避免使用索引来调用数据库中间层组件返回的结果集

6）对于自己创建的每一个类，都考虑置入一个 main()

其中包含了用于测试那个类的代码。 为使用一个项目中的类， 我们没必要删除测试代码。若进行了任何形式的改动， 可方便地返回测试。 这些代码也可作为如何使用类的一个示例使用。

7）应将方法设计成简要的、功能性单元

用它描述和实现一个不连续的类接口部分。理想情况下，方法应简明扼要。若长度很大， 可考虑通过某种方式将其分割成较短的几个方法。 这样做也便于类内代码的重复使用 （有些时候， 方法必须非常大， 但它们仍应只做同样的一件事情）。

8）设计一个类时，请设身处地为客户程序员考虑一下

（类的使用方法应该是非常明确的）。然后，再设身处地为管理代码的人考虑一下（预计有可能进行哪些形式的修改，想想用什么方法可把它们变得更简单）。

9）使类尽可能短小精悍，而且只解决一个特定的问题。

下面是对类设计的一些建议：一个复杂的开关语句：考虑采用 " 多形" 机制；数量众多的方法涉及到类型差别极大的操作： 考虑用几个类来分别实现；许多成员变量在特征上有很大的差别：考虑使用几个类。

10） 采用内部类

任何时候只要发现类与类之间结合得非常紧密，就需要考虑是否采用内部类，从而改善编码及维护工作。

11) 采用注释

尽可能细致地加上注释，并用 javadoc 注释文档语法生成自己的程序文档。

12）避免使用 " 魔术数字 "

这些数字很难与代码很好地配合。 如以后需要修改它， 无疑会成为一场噩梦 . 因为根本不知道 "100" 到底是指 " 数组大小 " 还是" 其他全然不同的东西 " 。所以， 我们应创建一个常数， 并为其使用具有说服力的描述性名称， 并在整个程序中都采用常数标识符。这样可使程序更易理解以及更易维护。

13）接口和抽象类的使用接口有如下特点：

1. 接口中的方法可以有参数列表和返回类型，但不能有任何方法体。

2. 接口中可以包含字段，但是会被隐式的声明为 static 和 final。

3. 接口中的字段只是被存储在该接口的静态存储区域内， 而不属于该接口。

4. 接口中的方法可以被声明为 public 或不声明，但结果都会按照 public类型处理。

5. 当实现一个接口时，需要将被定义的方法声明为 public 类型的，否则为默认访问类型， Java 编译器不允许这种情况。

6. 如果没有实现接口中所有方法，那么创建的仍然是一个接口。

7. 扩展一个接口来生成新的接口应使用关键字 extends ，实现一个接口使用implements 。接口和抽象类基本类似，具体选择则依据以下规则：

1. 如果要创建不带任何方法定义和成员变量的基类，那么就应该选择接口而不是抽象类。

2. 如果知道某个类应该是基类，那么第一个选择的应该是让它成为一个接口，只有在必须要有方法定义和成员变量的时候，才应该选择抽象类。因为 抽象类中允许存在一个或多个被具体实现的方法， 只要方法没有被全部实现该类就仍是抽象类。

14）日志调试

尽量减少程序使用System.out.println() 向 屏 幕 输 出 追踪的信息 ，System.out.println() 为单线程处理，因此在输出的时候，程序资源属于独占状态。采用日志控制进行信息的追踪，如log.debug,log.info,log.warn，不同的级别输出不同的信息，给用户带来的不同的提示。在系统开发阶段将日志级别设置为 DEBUG模式，而在项目运行期则设置为ERROR模式。