软件工程实验报告

软件编码规范

Java编码规范

1. 引言

软件开发涉及到各方面人员的交互、协作，为了有效地进行项目开发的沟通，完善代码的维护和交付，有必要在一个小组中采用统一的软件开发标准。一般来说，制定这样的标准有下列好处：

Ø 方便软件维护。据统计，80％的软件开发费用在维护，规范化的代码才方便维护，降低维护成本。

Ø 在软件的整个生命期内，期望一个编码人员从开始到该软件报废一致维护其代码是不现实的，必然需要不断地交付、协同

Ø 好的编码规范能够大大增强代码的可读性，便于开发人员快速的理解新代码。

Ø 任何产品都需要好的包装。我们可以把代码本身看作是一种产品，那么按照规范编程也是对这个“产品”的包装

Ø 规范化的代码也是软件质量的保证手段之一，也是软件过程能够流畅的基础。

2.文件

Ø 属性文件后缀为properties,并且符合java中i18n的规范；

Ø 对于各产品模块自己的配置文件必须放置在自己模块的conf目录下；

Ø 文件的命名需要有一定的意义，尽量简短，名称符合操作系统的要求且必须为英文或汉语拼音组成，文件名称不能存在空格，对于普通的文件单词与单词之间采用下滑线进行。

3.命名规则

3.1基本的规则

Ø 字符集在26个英文字母、0到9的阿拉伯数字和下划线之中。Java中类、字段、方法、变量、常量尽量用字母表达，没有特别的理由不能用任何的其他字符

Ø 命名需要有一定的意义，推荐采用问题域中的术语命名，使命名在一定程度上是自描述的

Ø 命名尽量的短，如果命名太长，可以采用别名的方式，或者缩写来简化命名。缩写一定要有的意义，而且需要在整个项目中维护这些缩写的意义

Ø 名称缩写的规则（对于类名、字段名、变量名称、模块名称等适用）

 1）删除所有的原音字母，压缩重复字母。如button,缩写为btn，

2）如发生命名冲突，则在某一缩写中保留原音。如batton,为了不与button冲突，缩写为batn

Ø 不要用前导下划线，也不要在命名的末尾用下划线

3.2常量命名

Ø 所有的字符都必须大写。采用有意义的单词组合表达，单词与单词之间以“\_”下划线隔开。

Ø 命名尽量简短，不要超过16个字符

程序开发中最好不要直接对literal进行工作，最好引入常量方式应用；只有在特别的情况下才能使用， 如在for循环中初始化变量时可直接用-1,0,1这些常量。

例：

public final int MAX\_SIZE = 120;

public final int MAX\_WIDTH = 100;

public final String PROPERTY\_NAME= "menu" ;

3.3变量命名

变量的命名包括实例变量，静态变量，函数参数的命名。

Ø 避免在命名中采用数字，除非命名意义明确，程序更加清晰，对实例变量的命名中不应该有数字

Ø 变量名称是名词意义

Ø 采用有符合问题域意义的单词或单词组合。第一个单词全部小写，后续的每个单词采用首字母大写，其余小写(特殊单词除外，如URL)

Ø 命名尽量简短，不要超过16个字符

Ø 除了生命周期很短的临时变量外，避免采用单字符作为变量名，实例变量的命名不要用单字符。常用的单字符变量如整型用 i、j、 k、 m、 n字符型用c、d、 e，坐标用x、y、z。

按钮

btnSubmit

Ø 不在特别的情况下，Java中不推荐采用前缀，而是推荐保持名称的语义

例：

public int width;

public String fileName;

public static ApplicationContext context;

3.4方法命名

命名多数为动词结构

Ø 采用有符合问题域意义的单词或单词组合。第一个单词采用小写，后续的每个单词采用首字母大写，其余小写（特殊字除外如URL），没有特别理由不用下划线作为分隔符

Ø 在Java中对属性方法命名遵循JavaBean的标准：

1） getter方法：　get+属性名, 对boolean型采用is+属性名，有些特定的属性名用has, can代替is可能更好

2） setter方法：　set+属性名

Ø 构造方法的命名与类名一致

如：

String getName();

string isStopped();

void connect();

3.5类和接口的命名

Ø 采用有符合问题域意义的单词或单词组合，每个单词的首字母大写，其余字母小写（特殊字除外如URL）

Ø 接口的第一个字符采用Ｉ

例：

public class Fiugre

public interface FiugreContainer

public class StdFigure //std为Standard的缩写

3.6包的命名

Ø 包名所有的字符都为小写；

Ø 两个不同业务的包之间不要双向依赖,可以单向依赖；

Ø 采用逻辑上的层次结构，从而减少依赖

4.注释规范

 4.1 基本规则

Ø 注释应该使代码更加清晰易懂

Ø 注释要简单明了，只要提供能够明确理解程序所必要的信息就可以了。如果注释太复杂说明程序需要修改调整，使设计更加合理。

Ø 注释不仅描述程序做了什么， 还要描述为什么要这样做,以及约束

Ø 对于一般的getter、setter方法不用注释

Ø 注释不能嵌套

Ø 生成开发文档的需要用中文编写

4.2 Java中有三种注释方式说明

4.2.1文档注释 /\*\* \*/

可以对用多行，一般用来对类、接口、成员方法、成员变量、静态字段、静态方法、常量进行说明。Javadoc可以用它来产生代码的文档。为了可读性，可以有缩进和格式控制。

文档注释常采用一些标签进行文档的特定用途描述,用于帮助Javadoc产生文档,常用的有：

标签

Used for

目的

@author name

类/接口

描述代码的作者，每个作者对应一个这样的标签

@deprecated

类

成员方法

说明该段API已经被废除

@exception name description

  或

  @throws name description

成员方法

描述方法抛出的异常

每个异常一个对应一个这样的标签

@param name description

成员方法

描述成员方法中的参数用途和意义，一个参数对应一个这样的标签

@return description

成员方法

描述成员方法的返回值的意义

@since

类/接口

成员方法

描述该段API开始的时间

@see ClassName

类/接口

成员方法

成员变量

用于引用特定的类描述，一般ClassName用包括包名的全名

@see ClassName#memberfunction

类/接口

成员方法

成员变量

用于引用特定的类的成员方法的描述，一般ClassName用包括包名的全名

@version text

类/接口

版本

@inheritDoc

类/接口

成员方法

继承的文档

4.2.2行注释 //

一次只能注释一行，一般用来简短的描述某一个局部变量，程序块的作用。

4.2.3块注释： /\* \*/

在代码中禁止使用。

4.3类/接口注释

类/接口描述，一般比较详细。按照常用的说明顺序排列，主要包括

Ø 类的主要说明，以。或.结束

Ø 类设计的目标，完成什么样的功能

Ø <Strong>主要的类使用</Strong>如何使用该类, 包括环境要求,如是否线程安全,并发性要求, 以及使用约束

Ø <Strong>已知的BUG</Strong>

Ø 描述类的修改历史:<Strong>修改人+日期＋简单说明</Strong>

Ø @author作者、@version版本, @see参照,@since开始版本等信息如：

/\*\*

 \* This class provides default implementations for the JFC <code>Action</code>

 \* interface. Standard behaviors like the get and set methods for

 \* <code>Action</code> object properties (icon, text, and enabled) are defined

 \* here. The developer need only subclass this abstract class and

 \* define the <code>actionPerformed</code> method.

 \* <p>

 \* <strong>Warning:</strong>

 \* Serialized objects of this class will not be compatible with

 \* future Swing releases.  The current serialization support is appropriate

 \* for short term storage or RMI between applications running the same

 \* version of Swing.  A future release of Swing will provide support for

 \* long term persistence.

 \* @version 1.41 02/02/00

 \* @author Georges Saab

 \* @see Action

 \*/

为了使形成的文档可读性好，注释中经常带有缩进和格式控制。类描述放在类的类定义的紧前面，不能有任何的空行。

4.4 变量注释

Ø 成员变量、类静态变量采用文档注释，对成员变量的注释通常包括：

1） 变量的意义

2） 变量的合法值域

3） 对并发访问的限制

如:

  /\*\*

\* Web.xml文件中configServlet参数的UIAPP.xml initparam

 \*/

    public final static String APP\_CONFIG = "aaa.uiapp";

Ø 局部变量，如算法相关的变量采用块或行注释

如：

void func() {

    int i; //用于循环计数

    …………

}

Ø 参数变量注释一般用文档注释,并且用@param来说明为参数,一般包括:

1） 参数的用途

2） 对参数值范围的要求

4.5 方法注释

描述函数的功能，对成员方法，静态方法一般采用文档描述，特别是公开的方法。注释可以很详细，为了可读性强也可包含格式控制，如下面说明含有缩进：

/\*\*

\* Here is a method comment with some very special

\* formatting that I want indent(1) to ignore.\*

\*/

方法注释一般包括:

Ø 方法的主要说明，以。或.结束

Ø 描述方法完成什么样的功能,方法的目标,用该方法的原因

 描述方法的使用方法,包括使用的环境要求,如前置条件,后置条件和并发性要求

描述已知的bug

 描述方法的修改历史：<Strong>修改人+日期＋简单说明</Strong> (<修改人+日期＋简单说明>)

param c elements to be inserted into this list.（参数说明）

return <tt>true</tt> if this list changed as a result of the call.(返回值说明)

throws    NullPointerException if the specified Collection is null.（异常说明）

see如果重载方法必须参考父类的方法

 Eclips下采用Alt+Shift+J生成Javadoc说明方法的放回值((@return)

4.6 修改记录

Ø 在修改一个类前，必须先从SVN中update，之后再进行修改；

Ø 修改的地方必须加入注释，说明修改人，修改原因，修改内容，修改时间；

5.编码规范

5.1基本原则

强制性原则：

Ø 字符串的拼加操作，必须使用StringBuilder；

Ø try…catch的用法，

try{

}catch{Exception e

e.printStackTrace();

}finally{

}//在最外层的Action中可以使用，其它地方一律禁止使用；

try{

//程序代码

}catch(Exception e){

//为空，什么都不写

}//在任何场景中都禁止使用

try{

}catch{Exception e

throw new runtimeException(e)；//最优先采用的写法

}finally{

}

Ø 对于捕获后，不知道干什么事情或者也不知道怎样处理的情况，就不要捕获异常，留给外出层去捕获处理；

Ø 返回类型为集合的,在方法声明中必须使用泛型，必须在javadoc中注明什么情况下返回null，什么情况下返回空集合。

Ø 对于方法、变量声明范围要采用如下优先级：private、protected、public，对于变量要采用如下的优先级：局部变量、实例变量、类变量，如果必须要采用实例变量或类变量的情况下，要保证线程安全性，如有可能尽量采用ThreadLocal保存实例变量或类变量；

Ø 如果不是必须，不要在循环中去定义变量或者new 对象；尽量在需要的最后一刻才去new 对象；

Ø 如果不是必须，不要在循环中去用try…catch;

Ø 类中对于比较复杂的逻辑要采用行注释的方式进行注释，java代码中绝对不允许采用块注释（/\*\*/）进行注释；

Ø Java类的名称第一个子母必须大写，有多个单词组成的，每个单词的首字母大写

Ø jsp的文件名必须全部小写；

Ø Spring的bean配置文件名必须小写，格式为xxx.bean.xml，xxx.bean.xml配置文件中的<bean  id=”” ，此处的id，就是将类名的第一个字母小写放到此处。

xwork的配置文件名必须小写，且遵循xwork\_xxx.xml的格式书写，其中XXX是业务名称的缩写；

 日志的处理，

if (log.isDebugEnabled())

ex.printStackTrace();

else

log.error("从数据库删除: [" + entity.getClass().getName() + "] 实例失败", daex);

throw new PersistenceException("从数据库删除: [" + entity.getClass().getName()

                    + "] 实例失败", daex);

代码中严禁使用System.out.println()进行调试输出，如果要使用调试信息，必须使用log.debug()。对于必要的信息使用log.info()进行输出；

类中不要出现无用import，可以采用IDE工具进行优化，类提交前进行代码的格式化；

有业务逻辑处理的类必须写junit单元测试类；

国际化的支持：ftl模板中不允许出现中文字符，要全部放到相应的properties文件中，properties文件要放到和Action类同样的目录中；ftl的编码要全部采用UTF-8的格式；properties文件的命名：中文：Action名称+“\_zh”+“\_CN”.properties，英文：Action名称+“\_en”+“\_US”.properties

一个程序文件最好不要超过２０００行

尽可能缩小对象的作用域，这样对象的可见范围和生存期也都会尽可能地小，尽所有可能优先采用局部变量，实在没有办法用全局变量的，优先采用ThreadLocal来处理。

一个方法所完成的功能要单一,不同的功能封装为不同的方法.

尽可能的处理异常或转换异常，不要一味的包装异常

如果对象在某个特定范围内必须被清理（而不是作为垃圾被回收），请使用带有finally子句的try块，在finally子句中进行清理。

对于把一些逻辑相关的类组织在一起，可以考虑把一个类的定义放在另一个类的定义中，这种情况推荐使用内部类（比如界面层中的事件响应等）。内部类拥有所有外围类所有成员的访问权。

对成员变量的访问最好通过getter/setter方法,这样能够保证访问的合法性,以及代码调整

优先选择接口而不是抽象类或具体类。如果你知道某些东西将成为基类，你应当优先把它们设计成接口；只有在必须放进方法定义或成员变量时，才把它修改为具体或抽象类。接口只和客户希望的动作有关（协议），而类则倾向于关注实现细节。

使用java标准库提供的容器。精通他们的用法，将极大地提高工作效率。优先选择ArrayList来处理顺序结构，选择HashSet来处理集合，选择HashMap来处理关联数组，选择linkedList来处理堆栈和队列，它对顺序访问进行了优化，向List中间插入与删除的开销小，但随机访问则较慢。当使用前三个的时候，应该把他们向上转型为List、Set和Map,这样就可以在必要的时候以其它方式实现

数组是一种效率最高的存储和随机访问对象引用序列的方式，但是当创建了一个数组对象，数组的大小就被固定了，如果在空间不足时再创建新的数组进行复制，这样效率就比ArrayList开销大了。所以必须明确使用场景。

 尽量使用”private”、”protected”关键字。一旦你把库的特征（包括类、方法、字段）标记为public,你就再也不可能去掉他们。在这种方式下，实现的变动对派生类造成的影响最小，在处理多线程问题的时候，保持私有性尤其重要，因为只有Private的字段才会受到保护，而不用担心被未受同步控制的使用所破坏。

禁止后台业务代码使用如下代码

try{

something()

}catch(Exception ex)

{}

new Exception()

5.2类编写规范

 类的结构组织，一般按照如下的顺序：

1． 常量声明

2． 静态变量声明

3． 成员变量声明

4． 构造函数部分

5． Finalize部分

6． 成员方法部分

7． 静态方法部分

这种顺序是推荐的，在实际开发中可以按照一定的尺度修改，原则是程序更易读。如对方法的排序按照重要性，或按照字母顺序排列或按照方法之间的关系排列。

每个方法（包括构造与finalize）都是一个段。多个变量声明按照逻辑共同组成一个段，段与段之间以空行分隔。

类声明时，要指出其访问控制，一般为没有修饰符，public，和private。

方法与方法之间，大的部分之间都需要以空行隔离。

编写通用性的类时，请遵守标准形式。包括定义equals()、hasCode()、toString()、Clone(实现Cloneable接口)，并实现Comparable和Serialiable接口

对于设计期间不需要继承的类，尽量使用final

5.3变量

对成员变量, 尽量采用private

每一个变量声明/定义占一行（参数变量除外），如：

int a;

int b;

比int a,b; 更容易读, 更容易查找bug

局部变量在使用前必须初始化，一般在声明时初始化

变量的声明要放在程序块的开始位置

如：

void myMethod() {

int int1 = 0; // beginning of method block

if (condition) {

int int2 = 0; // beginning of "if" block

...

}

}

一种例外情况是在for语句中，定义声明不仅不占一行，还在表达式内部，完全采用Eclips生成,如：

for(int i = 0; i<100; i++)

数组的申明采用 <数据类型[] + 变量名>方式如：

char[] buffer;

    而不是：

char buffer[];

5.4方法

对成员方法,不要轻易的采用public的成员变量。主要的修饰符有public, private, protected, 无

空方法中方法声明和函数体可都在一行。如：　void func(){}

方法和方法之间空一行

方法的文档注释放在方法的紧前面，不能空一行。

 避免过多的参数列表，尽量控制在５个以内，若需要传递多个参数时，当使用一个容纳这些参数的对象进行传递，以提高程序的可读性和可扩展性

方法中的循环潜套不能超过２层

对于设计期间不需要子类来重载的类，尽量使用final

每个方法尽量代码行数尽量不要超过100行（有效代码行，不包括注释），但必须保证逻辑的完整性

接口中的方法默认级别为protected，只有很确认其它子系统的包会调用自己子系统的接口中的方法时，才将方法暴露为public.

5.5 语言使用及书写规范

 避免变量的定义与上一层作用域的变量同名。

 方法与方法之间用需要用一空行隔开

局部变量在使用时刻声明，局部变量/静态变量在声明时同时初始化

在与常数作比较时常数放在比较表达式的前面如：

if(“simpleCase”.equals(obj))…

if(null == obj)….

return语句中，不要有复杂的运算。

switch语句，需要一个缺省的分支

6.框架相关规范

Spring配置文件中定义bean时，id或者name属性值的第一个单词首字母必须小写，且必须与java代码中声明的变量相同。xwork配置文件中定义action的name时，第一个单词首字母也必须要小写。

在spring bean或action的配置文件中声明的属性，在java代码中对应的属性必须要有getter和setter方法。