上海协同

< J2EE 项目>
项目开发规范

版本 <1.3>

修订历史记录

日期	版本	说明	作者
2006-05-08	1.0	初稿	蔡源
2006-06-28	1.1	修订	蔡源
2007-9-10	1.2	修订	蔡源
2007-12-9	1.3	修订	蔡源

1. 目录

项目开发规范

2. 简介

2.1 目的

本文试图提供一套编写高效可靠的 Java 代码的标准、约定和指导。它们以安全可靠的软件工程原则为基础，使代码易于理解、维护和增强。通过遵循和改进这些程序设计标准，使各项目产生的代码有更好的一致性，并提高软件开发团队的生产效率。

本规范适用于采用 J2EE 规范的项目，所有项目中的 Java 代码（含 JSP，SERVLET，JAVABEAN，EJB，开发工具生成的代码框架等）均应遵守这个规范。同时，也可作为其它项目的参考。

规范原则：

- ✓ 遵循业界标准
- ✓ 可读性强，意义清楚
- ✓ 整洁严谨、风格统一

这些规范的制定和执行有助于：

- ✓ 降低开发成本，减少项目开发周期，加速项目实施进度。
- ✓ 持续稳健的交付高质量的软件。
- ✓ 减轻开发人员的劳动强度和复杂度，后期维护和二次开发的劳动强度。
- ✓ 增强了团队合作，促进了团队成员的团队荣誉感。

2.2 适用人员

本文档的适用人员包括 J2EE 项目开发组全体成员：技术管理人员、系统设计人员、系统开发人员、系统测试人员、系统维护人员、推广培训人员及其他相关人员。

2.3 定义、首字母缩写词和缩略语

2.4 参考资料

- ■ 华为编程开发规范
- ■ 建行 Java 编码规范与指南
- ■ J2EE 完全参考手册

2.5 概述

3. 质量保证

- 在软件设计过程中构筑软件质量。
- 代码质量保证优先原则
  - ✓ 正确性，指程序要实现设计要求的功能。
  - ✓ 稳定性、安全性，指程序稳定、可靠、安全。
  - ✓ 可测试性，指程序要具有良好的可测试性。
  - ✓ 规范/可读性，指程序书写风格、命名规则等要符合规范。
  - ✓ 全局效率，指软件系统的整体效率。
  - ✓ 局部效率，指某个模块/子模块/函数的本身效率。
  - ✓ 系统的灵活性、适用性及可维护性原则。
  - ✓ 个人表达方式/个人方便性，指个人编程习惯。
- 防止引用已经释放的对象和资源。
- 模块独立性原则，避免重复定义，处理完整，输入输出明确
- 安全和保密原则
  - ✓ 系统应具有一定的容错能力，对一些错误事件（如用户误操作等）能进行自动补救。
  - ✓ 对一些具有危险性的操作代码（如写硬盘、删数据等）要仔细考虑，防止对数据、硬件等的安全构成危害，以提高系统的安全性。
- 认真处理程序所能遇到的各种出错情况。
- 系统运行之初，要初始化有关变量及运行环境，防止未经初始化的变量被引用。
- 系统运行之初，要对加载到系统中的数据进行一致性检查。
- 严禁随意更改其它模块或系统的有关设置和配置。
- 不能随意改变与其它模块的接口。
- 充分了解系统的接口之后，再使用系统提供的功能。
- 编程时，要防止差 1 错误。

  说明：此类错误一般是由于把“<=”误写成“<”或“>=”误写成“>”等造成的，由此引起的后果，很多情况下是很严重的，所以编程时，一定要在这些地方小心。当编完程序后，应对这些操作符进行彻底检查。
- 要时刻注意易混淆的操作符。当编完程序后，应从头至尾检查一遍这些操作符，以防止拼写错误。
- 使用第三方提供的软件开发工具包或控件时，要注意以下几点：
  - ✓ 充分了解应用接口、使用环境及使用时注意事项。
  - ✓ 不能过分相信其正确性。
  - ✓ 除非必要，不要使用不熟悉的第三方工具包与控件。

  说明：使用工具包与控件，可加快程序开发速度，节省时间，但使用之前一定对它有较充分的了解，同时第三方工具包与控件也有可能存在问题。

4. 命名规范

4.1 一般性规范

规范	是否必须遵守	示例
变量、函数采用匈牙利命名法。命名中若使用特殊约定或缩写，则要有注释说明。**严禁使用拼音或拼音缩写命名！**	必须	
备注		

4.2 常量

规范	是否必须遵守	示例
全部大写，不同单词间用下划线分隔	必须	`private static final String LIST_SQL = “SELECT * FROM TABLE”;`
备注		

4.3 类

规范	是否必须遵守	示例
类名首字母应该大写。对于所有标识符，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。	必须	ThisIsAClassName
备注		
遗留系统等特殊除外		

4.4 接口

规范	是否必须遵守	示例
以大写“I”字母开头，后同实现类的名称。	必须	IThisIsAInterface
备注		

4.5 字段和方法

规范	是否必须遵守	示例
字段、方法以及对象（句柄）的首字母应小写。对于所有标识符，其中包	必须	thisIsMethodOrFieldName

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。		

采用动词+名词的命名方式	必须	addNews();
		updateNews();
		deleteNews();
		listNews();

类的布尔型的判断方法要求方法名使用单词 is 做前缀或者使用具有逻辑意义的单词	必须	Boolean isMan(); Boolean isString();
备注		

4.6 数组

规范	是否必须遵守	示例
数组的声明必须定义在类型上	必须	正确：byte[] buffer 错误：byte buffer[]
备注		

4.7 JSP

规范	是否必须遵守	示例
采用采用名词+动词的命名方式，完整的英文描述说明 JSP 所完成的功能，不同单词间用下划线分隔	必须	正确：operator_add.jsp 错误：addoperator.jsp
备注		

5. 一般性规范

规范	是否必须遵守	示例
总是检查 NULL，绝对不要出现 NullPointerException 的例外	必须	

在一个特定的作用域内，若一个对象必须清除（非由垃圾收集机制处理），请采用下述方法：初始化对象；若成功，则立即进入一个含有 finally 从句的 try 块，开始清除工作	必须	try
		{
		//数据库操作
		doSomething();
		}
		catch(Exception e)
		{
		//捕获并处理异常
		}
		finally
		{
		try
		{
		//关闭数据库连接
		conn.close();
		}
		catch(Exception e)
		{
		//记录异常
		}
		}

比较的时候要把常量放在操作符左边	建议	if(“0”.equals(abc)) {}
类中用 this 关键字来访问同一个类中的非静态方法或变量	建议	
进行字符串连接的时候，尽量使用 StringBuffer 类来进行类似处理	建议	
一旦某个对象不再使用，尽早置 NULL	建议	
不要连续的进行对象创建，特别是在循环中	建议	
明确的初始化一个构造类里面的所有的字段	建议	
让一切东西都尽可能地"私有"- private。若只公布自己必须公布的，就可放心大胆地改变其他任何东西。在多线程环境中，隐私是特别重要的一个因素，只有 private 字段才能在非同步使用的情况下受到保护。	建议	
备注		

6. 格式规范

注：格式规范的目的是为了提高程序的可阅读性，没有强制的一定的规范；但由于个人开发习惯和经验的不同，会影响项目团队成员和相互协作，固提出以下规范。同时配合开发工作的代码格式化模板，轻松统一整个系统代码格式。

6.1 缩进

格式规范	是否必须遵守	示例
子功能块应在其父功能块后缩进	必须	
功能块中缩进过深时应将子功能提取出来作为子内部方法	必须	
使用一个 tab 作为缩进的单元	必须	

6.2 空行

格式规范	是否必须遵守	示例
两个方法之间使用一个空行	必须	
方法内的局部变量和方法的第一条语句之间使用一个空行	必须	
块注释或单行注释之前使用一个空行	必须	
一个方法内的两个逻辑段之间使用一个空行	必须	
备注		
空行将逻辑相关的代码段分隔开，以提高可读性。		

6.3 行长度

格式规范	是否必须遵守	示例
单个方法的代码行(不包含注释行)不允许超过 100 行	必须	
单个类的代码行(包含注释行)不允许超过 1500 行	必须	
一行的长度尽量避免超过 80 个字符，因为很多的终端和工具不能很好的处理，可能导致无法正确显示	必须	
备注		
1. 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。 2. 单个函数的代码行超过 100 行时可以使用子函数等将相应功能抽取出来；单个类的代码行超过 1500 行时可以将相应功能的代码重构到其它类中；长语句行宽超过 80 列时可以在逗号后或操作符前折行，并比原语句再缩进 4 个空格。		

7. 表达式和语句规范

7.1 简单语句

格式规范	是否必须遵守	示例
每一行只能包含一个语句	必须	i++; j++;
备注		

7.2 条件语句

格式规范	是否必须遵守	示例

在混合运算表达式中使用括号避免运算优先级问题，即使对运算的优先顺序非常清晰，也应该这么做。(同时适用于 while 语句内条件说明)	建议	不可用以下方式:
		if (a == b && c == d)
		用以下方式:
		if ((a == b) && (c == d))

备注		

7.3 分支语句

格式规范	是否必须遵守	示例

每一个 switch 的 case 必须有 break 语句，并且必须有 default 做为最后出口	必须	switch (x) {
		case 1 : {
		expr;**break;**
		} // end case
		case 2 : {
		expr;**break;**
		} // end case
		default : {
		expr;**break;**
		} // end default
		} // end switch

备注		

7.4 循环语句

格式规范	是否必须遵守	示例
While 和 Do-While 的使用要注意循环次数的区分。		While 先判断再执行 While(condition){…} Do-While 先执行再判断 Do{…}While(condition)
备注		

7.5 声明语句

格式规范	是否必须遵守	示例
每一行**只能**包含一个语句	必须	i++; j++;

避免声明的局部变量覆盖上一级声明的变量。	必须	int count;
		...
		myMethod() {
		if (condition) {
		int count = 0;
		...
		}
		...
		}

推荐一行写一个声明，因为这样以利于写注释。	建议	int a; //a 的注释
		int b; //b 的注释

尽量在声明局部变量的同时初始化。唯一不这么做的理由是变量的初始值依赖于某些先前发生的计算。	建议	
备注		

7.6 异常处理语句

格式规范	是否必须遵守	示例
不要有空的 catch 语句出现，至少打印出在哪里抛出例外和相关信息	必须	catch(IOException e) {};
注意 Finally 的使用：无论是否抛出例外，都会执行 Finally 块中的语句		
备注		

7.7 类和接口

格式规范	是否必须遵守	示例
类、接口定义之前应先进行注释。注释包括类、接口的目的、作用、功能、继承的父类，实现的接口、实现的算法、使用方法、示例程序等，还可以包括期望改进工作的地方和不希望改变的地方	必须	
备注		

7.8 常量

规范	是否必须遵守	示例
若在定义中出现了常数初始化字符，则大写 static final 基本类型标识符中的所有字母。这样便可标志出它们属于编译期的常数。	必须	Private static final String LIST_SQL = “SELECT * FROM TABLE”;
备注		

7.9 方法

格式规范	是否必须遵守	示例
声明原则是尽量保证私有性：如非必须 public 则 protected 如非必须 protected 则 private	建议	

应验证所有传入参数，不能假定非空指针，验证有错误时返回明确的错误信息	建议	If(Parameter1!=NULL){
		…}
		// endIf
		if(Parameter2!=NULL) {
		…}
		// endIf

尽量保证每个方法只有一个出口，否则可能出现丢失返回的情况	建议	boolean check() {
		  boolean result = false;
		  if (...) {
		  result = true;
		  }
		  return result;
		}

备注		

7.10 属性

格式规范	是否必须遵守	示例
数字常量不要直接用在代码中，除-1，0，1 外，它们可也出现在 for 循环中作为计数值	必须	
在子类中不要重复定义父类中的成员变量	必须	
不要出现 public 的成员变量，原则上都是 private，用 getter 和 setter 进行访问，保证封装性	建议	
备注		

8. 日志规范

规范	是否必须遵守	示例
在正式提交的代码中，不允许使用 System.out 方式打印信息	必须	
在程序发生错误，抛出异常等时应记录日志信息	必须	
日志应区分如下几类情形： ■ 发生致命错误时运行将终止(Fatal) ■ 发生错误仍可继续执行(Error) ■ 发生警告仍可继续执行(Warn) ■ 记录运行过程或干预情况(Info) ■ 调试程序记录信息等(Debug)	必须	

备注

- 致命(Fatal)

  严重的错误，系统无法正常运行，如硬盘空间满等。这个级别很少被用，常暗含系统或者系统的组件迫近崩溃。

- 错误(Error)

  系统可以继续运行，但最好要尽快修复的错误。这个级别用的较多，常常伴随 Java 异常，错误(Error)的环境不一定会造成系统的崩溃，系统可以继续服务接下来的请求。

- 警告(Warn)

  系统可以正常运行，但需要引起注意的警告信息。这个级别预示较小的问题，由系统外部的因素造成的，比如用户输入了不符合条件的参数。

- 信息(Info)

  系统运行的主要关键时点的操作信息，一般用于记录业务日志。但同时，也应该有足够的信息以保证可以记录再现缺陷的路径。这个级别记录了系统日常运转中有意义的事件。

- 调试(Debug)

  系统运行中的调试信息，便于开发人员进行错误分析和修正，一般用于程序日志，关心程序操作(细粒度)，不太关心业务操作(粗粒度)

9. 注释规范

规范	是否必须遵守	示例
类注释：类的功能、接口定义，创建类的作者和时间	必须	
方法注释：必须明确方法、出入函数、返回值、异常的含义	必须	/** * 这里是方法的含义. * @param opCode 操作员编号 * @return serialNo 序号 */
修改其他人写的文件时请署上您的大名和时间。	必须	/** * 这里是方法的含义. * @param opCode 操作员编号 * @return serialNo 序号 * * update by caiyuan 2007-9-11 */
尽可能细致地加上注释	建议	

备注

1. 注释要简单明了。
2. 编写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。
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. 全局变量要有较详细的注释，包括对其功能、取值范围、哪些函数或过程存取它，以及存取时注意事项等的说明。
5. 在每个函数或过程的前面要有必要的注释信息，包括：函数或过程名称；功能描述；输入、输出及返回值说明；调用关系及被调用关系说明等。

【注释的要素项】

- ✓ 方法如何处理错误条件或不合要求的输入
- ✓ 如何将错误条件传回给调用者
- ✓ 可能会抛出哪个特定异常的子类
- ✓ 哪些值对于输入是有效的
- ✓ 类不变条件、方法前置条件或方法后置条件
- ✓ 副作用
- ✓ 在方法之间是否有重要联接
- ✓ 类如何处理多个线程同时访问一个实例的情况。

请使用统一的注释风格，示例如下：

Test.java

```java
package singlee.intrust.audit;

/**
 * <b>标题:</b> JAVA DOC演示.
 * <pre><b>描述:</b>
 *   这里演示JAVA DOC的书写
 *   这里是第二行
 *   这里是第三行
 *   这里是第四行
 * </pre>
 *
 * @author <a href="mailto:littcai@hotmail.com">空心大白菜</a>
 * @since 2004-8-2
 * @version 1.0
 */
public class Test {

    /**
     * 构造函数的注释.
     */
    public Test() {};

    /**
     *
     * 该方法的用途.
     *
     * @param a 参数a的含义
     */
    public void dosomething(String a) {}

    /**
     *
     * 该方法的用途.
     *
     * @param a 参数a的含义
     * @return 返回值
     * @throws Exception
     *
     * @author 孙悟空 2008-10-4
     */
    public String dosomething2(String a) throws Exception
    {
        return a;
    }

}
```

10. 性能规范

规范	是否必须遵守	示例
避免在循环中频繁构建和释放对象。不再使用的对象应及时释放，如无必要，不要序列化对象	必须	
不需要重新赋值的变量(含类变量、实例变量、局部变量)声明成 final，可提高程序响应效率	必须	
在处理 String 的时候（如多个字符串的累加）要使用 StringBuffer 类，处理完成后将 StringBuffer 对象再转换为需要的 String 对象	必须	
避免使用太多的 synchronized 关键字，必须使用时，也应尽量控制范围，最好是块级控制	必须	
与数据库连接的占用时间尽可能地短，数据库连接采用连接池，并及时关闭	必须	
尽量使用来自 Java API 的类，因为它们本身已针对机器的性能进行了优化	必须	
对不正常的条件使用异常，尽可能准确的使用预定义的异常	必须	
对频繁使用而实时性不强的数据采用适当的 Cache 机制，并使用增量 Cache 及 TimeOut	建议	
备注		

11. 错误处理和异常事件规范

规范	是否必须遵守	示例

尽量采用含义明确的异常,如 SQLException、NotLoginException、ParamNotSetException；而避免使用大而空的 Exception，这样有助于异常发生之后更精确的定位错误的类型、位置，以及针对这些异常的后续处理。	必须	try
		{
		doSomeSqlOperatorion();
		doSomeLoginedOperation();
		}
		catch(SQLException e)
		{
		//这里捕获的是 SQL 异常
		}
		catch(NotLoginException e)
		{
		//这里捕获的是未登陆异常
		}

当发生一个异常时，尤其这个异常调用了某些资源，要主动的以编程方式释放这些资源	必须	try
		{
		doSomeSqlOperatorion();
		}
		catch(SQLException e)
		{
		//这里捕获的是 SQL 异常
		}
		finally
		{
		conn.close();//关闭数据库连接
		}

备注		

12. JSP规范

规范	是否必须遵守	示例
在 JSP 中避免代码重复。把要重复的功能放在一个包含的 jsp、bean 或标记扩展中，使得它能够被重用	必须	
JSP 层不应该直接访问数据，这包括 JDBC 数据库访问和 EJB 访问	必须	
应当使用隐藏的注释来阻止输出的 HTML 过大	必须	
在 JSP 中避免进行异常处理	建议	
备注		

13. 界面规范

规范	是否必须遵守	示例
防止表单重复提交处理	必须	对提交按钮点击后做变灰处理避免在网络响应较慢情况下用户重复提交同一个表单。使用页面过期失效避免用户后退浏览重复提交表单。
尽可能早的在客户端完成输入数据合法性验证	必须	
鼠标动作提示和回应	建议	对用户的鼠标定位操作，当移动到可响应的位置上时，应给予视觉或听觉的提示
输入控件的自动聚焦和可用键盘切换输入焦点	建议	

14. 版本控制

规范	是否必须遵守	示例
离开公司之前 Commit 所有文件，上班之后 Update 所有文件，并根据 Daily build 的报告，修正昨天提交的版本中的错误信息。	必须	
在修改文件之前，先 Update 一下，使得修改时的版本尽可能新，一旦发生冲突，解决它的工作量会比较小。如果是大家共同使用，并经常修改的文件，如 Constants.java、struts-config.xml 等文件，更要如此，最好能通知其他人员 Commit。	必须	
在文件出现冲突时，应该先进行比对，将服务器端的修改更新至本地，不清楚的地方要同其他人沟通，合并完成后，选择 Override and Commit 更新，禁止不进行比对就进行 Override and Commit 操作！	必须	
提交的文件必须经过测试，起码保证在本地是可以编译通过的，方便其他人测试。	必须	
对二进制文件，如 Word 文档，CVS 不能进行比对，如果出现冲突，需要自己手工合并，可以将本地文件备份，将文件 Override and Update，然后从备份文件中将自己修改的部分合并之后提交。	必须	
CVS 不允许进行删除服务器端文件，所以大家最好不要提交临时文件及临时目录，如编译产生的文件和目录。	必须	
不同功能模块的文件分开 Commit，同一功能模块的文件一次 Commit，对于分析错误、减少冲突、版本回退等有帮助。	建议	
备注		
在 Eclipse 同 CVS 服务器同步之前，要先刷新，否则可能会因为在 Eclipse 外面编辑的文件，Eclipse 中没有刷新而导致同步报错，可以将 Eclipse 设置为自动刷新，操作方法：Window->Preferences->Workbench，选中 Refresh workspace automatically。即使如此，仍然可能出现问题，因为 Eclipse 刷新可能会有一定延迟，所以在报错时，手动刷新一下，再同步。		

15. 代码测试和发布

规范	是否必须遵守	示例
代码版本升级要经过严格测试	必须	
清理、整理或优化后的代码要经过审查及测试	必须	
正式版本上软件的任何修改都应有详细的文档记录	必须	
使用工具软件对代码版本进行维护	建议	
单元测试要求至少达到语句覆盖	建议	

16. 附录：指南概要

16.1 附录 1：质量评审标准

16.1.1 评审目标

本规范制定同时用于系统编程阶段的质量评审依据。项目组可根据团队情况作具体要求或完善。

质量评审采用代码走查。

16.1.2 扣分标准

编程评审按百分制进行倒扣分，最后按权重折成编码实际得分，编程百分制分为两项：

- 基本分：60 分
- 规范分：40 分

具体评审扣分标准如下：

- 完成测试验收的，得基本分(60 分)。
- 未能完成测试验收的，视质量好坏得分（较为齐全的：50~59，基本齐全的：40~49，较为欠缺的：0~39）。
- 同一开发者每发现一项不符合规范扣 3 分。
- 不同开发者发现同类一项不规范扣 5 分，与上一项不重复扣分。
- 代码重用性差(含未使用公共函数)，一次性扣 10 分。
- 规范分扣完为止。

16.2 附录 2：有效地使用这些标准

以下的建议将帮助你更有效地使用本文所描述的 Java 编程规范和指导：

1. **理解标准。**花些时间去理解为什么每个标准和指导会使开发效率提高。比如说，不要仅仅是因为指导中要求你才在一行仅声明一个局部变量，而应该是因为你明白它能使你的代码更易懂你才这样做。

2. **信任这些标准。**理解每个标准是一个开始，但你还需要信任这些标准。遵守标准不应仅仅是当你有时间才做的事，而你应该一直遵守，因为你相信这是最好的程序设计方法。

3. **当你写代码时就应该遵守标准，而不应是一个事后的想法。**加了注释的代码不仅在你写程序时，而且在你写完程序时，都更容易理解。在程序开发阶段和维护阶段，一致性地命名成员函数和字段都使工作更加容易。在开发和维护阶段，整洁的代码让工作更加容易。概括起来说，遵守标准将提高你开发过程中的生产率，并且使你的代码更易维护（因此也使维护者的生产率提高了）。如果从一开始你就写出整洁的代码，你将在撰写过程中受益。

4. **使它们成为你的质量保证的过程。**代码检查的一部分应该是确保源码遵守你的机构所采用的标准。将标准作为你训练和指导开发人员更有效率的基础。