**软件编码规范说明书**

**第一版**

**西北师范大学**

**你说对就队开发组**

**目 录**

[第一章 规范说明 3](#_Toc22044)

[1.1 目的 3](#_Toc683)

[1.2 范围 3](#_Toc6632)

[第二章 注释规范 3](#_Toc8229)

[2.1 实现注释的格式 3](#_Toc384)

[2.2 块注释 3](#_Toc14701)

[2.3 单行注释 4](#_Toc5401)

[2.4 尾端注释 4](#_Toc29038)

[2.5 行末注释 5](#_Toc24918)

[2.6 文档注释 5](#_Toc18964)

[第三章 命名规范 5](#_Toc20707)

[3.1 变量（Variable）命名 5](#_Toc31246)

[3.2 常量命名 6](#_Toc953)

[3.3 类（Class）命名 6](#_Toc23908)

[3.4 接口（Interface）命名 7](#_Toc6605)

[3.5 方法（Method）命名 7](#_Toc16388)

[3.6 名称空间(Namespace)命名 7](#_Toc21499)

[第四章 编码规则 7](#_Toc31092)

[4.1 错误检查规则 7](#_Toc17760)

[4.2 大括号规则 8](#_Toc19269)

[4.3 缩进规则 8](#_Toc24715)

[4.4 小括号规则 8](#_Toc18219)

[第五章 总结 9](#_Toc30386)

# 规范说明

## 目的

* 为了整合开发团队的小组成员在开发设计过程的编程规范；
* 是开发团队的每个小组成员理解说明中的每一条规范；
* 为了方便开发组的小组成员能相互读懂对方的代码，从而使每个人的代码风格一致，约束条件统一；
* 编码规范有助于提高代码的可读性，有助于代码的后期迭代适用于基于Java语言开发的代码；
* 一个软件的生命周期中大部分时间都在维护代码，所以遵循相应的代码规范对于后期维护很重要；

## 范围

本规范适用于开发团队的全体人员，作用于软件项目开发的代码编写阶段和后期维护阶段。

# 注释规范

## 实现注释的格式

编码规范中注释块的形式有4中：块(Block)，单行(single-line)，尾端(trailing)和行末(end-of-line)。

## 块注释

块注释通常用于提供对文件，方法，数据结构和算法的描述。块注释被置于每个文件的开始处以及每个方法之前。它们也可以被用于其他地方，比如方法的内部。在功能和方法内部的块注释应该和它们所描述的代码具有一样的缩进格式。

块注释之首应该有一个空行，用于把块注释和代码分割开来，比如：

/ \*

\* zhe shi yi ge zhu shi.

\*/

块注释可以以/ \*-开头，这样indent(1)就可以将之识别为一个代码块的开始，而不会重排它。

/ \*-

\* zhe shi yi ge yuju l

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

\*

\* one

\* two

\* three

\*/

注意：如果你不使用indent(1)，就不必在代码中使用/ \*-， 或为他人可能对你的代码运行indent(1)让步。

## 单行注释

短注释可以显示一行内，并与其后的代码具有一样的缩进层级。如果一个注释不能在一行内写完，就该块注释。单行注释之前应该有一个空行。以下是一个Java代码中单行注释的例子：

if (condition) {

System.out.println(“”); / \* 单行注释 \*/

……

}

## 尾端注释

极短的注释可以与它们所要描述的代码位于同一行，但是应该有足够的空白来分开代码和注释。若有多个短注释出现于大段代码中，它们应该具有相同的缩进。

if (v<=0)

{

return true; / \* 正确情况 \*/

} else {

return false;); / \* 失败情况 \*/

}

## 行末注释

注释界定符“//”，可以注释掉整行或者一行中的一部分。它一般不用于连续多行的注释文本；然而，它可以用来注释掉多行的代码段。这样可以避免我们即不想现在使用这些代码的问题，又可以在我们想要用的时候再重新使用它们，如下所示：

// if(a>0){

// System.out.println();

// }

## 文档注释

* 文档注释可以在Java中生成帮助文档，以/\*\*开头，以\*/结尾；
* 每个注释包含若干个文本及javadoc标签
* Javadoc标签以@开头为前缀，
* 常见javadoc标签；

|  |  |
| --- | --- |
| @author | 作者名 |
| @parameter | 参数及其含义 |
| @return | 返回值 |
| @version | 版本 |
| @since | 最早使用该方法、类、接口的jdk版本 |
| @throws | 异常类及抛出条件 |
|  |  |

# 命名规范

## 变量（Variable）命名

程序中变量名等于变量的前缀和代表变量含意的英文单词或单词的开头缩写或者英文单词的每个部分的首字母组成，如下所示。

private string m\_strName;

private DateTime m\_dtDate;

string stuName；

string stuBanji;

## 常量命名

常量名也应当有一定的意义，格式为 NOUN 或 NOUN\_VERB。常量名均为大写，字之间用下划线分隔。

例：

private const bool WEB\_ENABLEPAGECACHE\_DEFAULT = true;

private const int WEB\_PAGECACHEEXPIRESINSECONDS\_DEFAULT = 3600;

private const bool WEB\_ENABLESSL\_DEFAULT = false;

注：

变量名和常量名最多可以包含 255 个字符，但是，超过 25 到 30 个字符的名称比较笨拙。此外，要想取一个有实际意义的名称，清楚地表达变量或常量的用途，25 或 30 个字符应当足够了。

## 类（Class）命名

* 类的名称应该凸显事物的特征；
* 名字一般使用全称，除非他的简写是大家公认的；
* 名字可以有两个或三个单词组成，不应太多；
* 在名字中，所有单词第一个字母大写，比如User；
* 尽量使用名词或名词短语来进行类的命名；
* 不要使用下划线；

## 接口（Interface）命名

其余规则和类命名规则相同，唯一的区别是接口命名是在前面加上一个前缀“I”。如下所示。

interface IJiekou;

## 方法（Method）命名

和类命名规范相同，要让人一目了然，具有极高辨识度，不能起一些只有开发人员自己看懂，而别人看不懂的方法名。

## 名称空间(Namespace)命名

和上面规范一样。

# 编码规则

## 错误检查规则

* 编程中要考虑函数的各种执行情况，尽可能处理流程中碰到的各种情况。
* 对系统调用的所有错误信息进行检查。
* 将函数分两类：一类为与屏幕的显示无关， 另一类与屏幕的显示有关。对于与屏幕显示无关的函数，函数通过返回值来报告错误。对于与屏幕显示有关的函数，函数要负责向用户发出警告，并进行错误处理。
* 错误处理代码一般放在函数末尾。
* 对于通用的错误处理，可建立通用的错误处理函数，处理常见的通用的错误。

## 大括号规则

代码中大括号通常另起一行，这样方便维护代码。如下所示：

if （a>0）

{

System.out.println( );

}

## 缩进规则

使用一个“Tab”为每层次缩进。

## 小括号规则

* 不要把小括号和关键词（if 、while等）紧贴在一起，要用空格隔开它们。
* 不要把小括号和函数名紧贴在一起。
* 除非必要，不要在Return返回语句中使用小括号。因为关键字不是函数，如果小括号紧贴着函数名和关键字，二者很容易被看成是一体的。

# 总结

目前，本开发组的代码编写规范暂时就这些，其中借鉴了一些成熟的团队的代码编写规范，后期开发系统时，有需要添加的会继续更新，并根据情况作出规范的变动。