软件工程管理

软件需求工程

软件需求工程-住宅维护系统

系统编码与实现计划

组　　号： G09

组　　长： 江 号

组　　员： 沈子衿 宋宇杰 王优

林宇翔 徐正杰

2018 年 12 月 14 日

修改历史

日期	版本	作者	修改内容
2018.12.14	1.0	江号、沈子衿、宋宇杰、王优、林宇翔、徐正杰	初稿

目录

1. 引言

1.1. 编写目的

该文档的目是描述软件工程专业课该文档的目是描述软件工程专业课——住宅维护系统的目的编码规范和对代码的说明。为了更好地提高技术人员的工作效率，保证开发的有性和合理性，并可最大程度地提高序代码的可读性和可重复利用性，制定此规范。项目小组根据实际情况，可以对本规范进行补充或裁减。

其主要内容包括：

1. 编码规范
2. 安全规范
3. 命名规范
4. 注释规范
5. 语句规范
6. 声明规范
7. 目录设置
8. 代码说明

该文档的预期读者是：

9. 开发人员
10. 项目管理人员

1.2. 范围

该文档定义了本项目的代码编写规范，以及部分代码描述和所有代码的说明。

1.3. 缩写说明

无

1.4. 术语定义

无

1.5. 版本更新信息

修改编号	修改日期	修改后版本	修改位置	修改内容概述
001	2018.12.14	1.0	全部	初始发布版本

2. 安全规范

2.1. 保护规则

2.1.1. 命名规则

Java 程序应用下列文件后缀：

文件类别	文件后缀
Java 源文件	.java
Java 字节码文件	.class

2.1.2. 存放规则

一般包含文件不需要直接暴露给用户，所以应该放在 Web Server 访问不到的目录，避免因为配置问题而泄露设置信息

2.2. 安全规则

请参考产品安全检查表。

1.输入和输出检查是否做了 HTML 代码的过滤

可能出现的问题：如果有人输入恶意的 HTML 代码，会导致窃取 cookie,产生恶意登录表单，和破坏网站

2.检查变量做数据库操作之前是否做了 escape

可能出现的问题：如果一个要写入查询语句的字符串变量包含了某些特殊的字符，比如引号(',")或者分号(;)，可能造成执行了预期之外的操作。

建议采用的方法：使用 mysql_escape_string()或实现类似功能的函数。检查输入数值的合法性

3.异常的数值

可能出现的问题：异常的数值会造成问题。如果对输入的数值不做检查会造成不合法的或者错误的数据存入 UDB、存入其它的数据库或者导致意料之外的程序操作发生。

举例：

如果程序以用户输入的参数值做为文件名，进行文件操作，恶意输入系统文件名会造成系统损毁。

4.核实对 cookie 的使用以及对用户数据的处理

可能出现的问题：不正确的 cookie 使用可能造成用户数据泄漏

5.访问控制

对内部使用的产品或者供合作方使用的产品，要考虑增加访问控制 logs

确保用户的保密信息没有记在 log 中(例如：用户的密码)

确保对关键的用户操作保存了完整的用户访问记录

6.https

对敏感数据的传输要采用 https，不能使用 http 协议

2.3. 一些针对 Java 的规则

每个 Java 源文件都包含一个单一的公共类或接口。若私有类和接口与一个公共类相关联，可以将它们和公共类放入同一个源文件。公共类必须是这个文件中的第一个类或接口。

Java 源文件还遵循以下规则：

-开头注释（参见"开头注释"）

-包和引入语句（参见"包和引入语句"）

-类和接口声明（参见"类和接口声明"）

2.4. 一些其他处理规则

2.4.1. 对输入参数值进行转义处理

页面接到参数需要 SQL 操作，这时候需要做转义，尤其需要注意";"。如：a="Let's Go"; sql="Insert into tmp(col) values('a')";这种情况出现错误的不确定性。

2.4.2. 操作 HTML 文本

很多时候需要存放一大段 HTML 文本供页面使用，象用户定制页头页脚等。需要剔除脚本标记，避免执行恶意代码。

转换”<"">“号，保证代码完整。

3. 命名规则

制定统一的命名规范对于项目开发来说非常重要，不但可以养成程序员一个良好的开发

习惯，还能增加程序的可读性、可移植性和可重用性，还能很好的提高项目开发的效率。

3.1. 变量命名

变量命名分为普通变量、静态变量、局部变量、全局变量、Session 变量等方面的命名规则。

3.1.1. 普通变量

普通变量命名遵循以下规则：

a．所有字母都使用小写；

b．对于一个变量使用多个单词的，使用'_'作为每个词的间隔。

例如：base_dir、red_rose_price 等

3.1.2. 静态变量

静态变量命名遵循以下规则：

a．静态变量使用小写的 s_开头；

b．静态变量所有字母都使用小写；

c．多个单词组成的变量名使用'_'作为每个词的间隔。

例子：s_base_dir、s_red_rose_prise 等。

3.1.3. 局部变量

局部变量命名遵循以下规则：

a．所有字母使用小写；

b．变量使用'_'开头；

c．多个单词组成的局部变量名使用'_'作为每个词间的间隔。

例子：_base_dir、_red_rose_price 等。

3.1.4. 全局变量

全局变量应该带前缀'g'，知道一个变量的作用域是非常重要的。

例如：

global gLOG_LEVEL;

global gLOG_PATH;

3.1.5. 全局常量

全局变量命名遵循以下规则：

a．所有字母使用大写

b．全局变量多个单词间使用'_'作为间隔。

例子：BASE_DIR、RED_ROSE_PRICE 等。

3.1.6. session 变量

session 变量命名遵循以下规则：

a．所有字母使用大写；

b．session 变量名使用'S_'开头；

c．多个单词间使用'_'间隔。

例子：S_BASE_DIR、S_RED_ROSE_PRICE 等。

3.2. 类

3.2.1. 类

Java 中类命名遵循以下规则：

a．以大写字母开头；

b．多个单词组成的变量名，单词之间不用间隔，各个单词首字母大写。

例子：class MyClass 或 class DbOracle 等。

3.2.2. 方法或函数

方法或函数命名遵循以下规则：

a．首字母小写；

b．多个单词间不使用间隔，除第一个单词外，其他单词首字母大写。

例子：function myFunction ()或 function myDbOracle()等。

3.2.3. 缩写词

当变量名或者其他命名中遇到缩写词时，参照具体的命名规则，而不采用缩写词原来的全部大写的方式。

例子：function myPear（不是 myPEAR） functio getHtmlSource（不是 getHTMLSource）。

3.3. 数据库表名

数据库表名命名遵循以下规范：

a．表名均使用小写字母；

b．对于普通数据表，使用_t 结尾；

c．对于视图，使用_v 结尾；

d．对于多个单词组成的表名，使用_间隔；

例子：user_info_t 和 book_store_v 等

3.3.1. 数据库字段

数据库字段命名遵循以下规范：

a．全部使用小写；

b．多个单词间使用_间隔。

例子：user_name、rose_price 等。

4. 书写规范

书写规则是指在编写 Java 程序时，代码书写的规则，包括缩进、结构控制等方面规范：

4.1. 程序编写

4.1.1. 代码缩进

在书写代码的时候，必须注意代码的缩进规则，我们规定代码缩进规则如下：

a．使用 4 个空格作为缩进，而不使用 tab 缩进（对于 ultraedit，可以进行预先设置）

例子：

```
for(i=0;i<count;i++)
{
    echo "test";
}
```

4.1.2. 大括号{}书写习惯

在程序中进行结构控制代码编写，如 if、for、while、switch 等结构，大括号传统的有两种书写习惯，分别如下：

a．{直接跟在控制语句之后，不换行，如

```
for(i=0;i<count;i++){
    echo "test";
}
```

b．{在控制语句下一行，如

```
for(i=0;i<count;i++)
{
    echo "test";
}
```

其中，a 是 PEAR 建议的方式，但是从实际书写中来讲，这并不影响程序的规范和影响用 Javadoc 实现文档，所以可以根据个人习惯来采用上面的两种方式，但是要求在同一个程序中，只使用其中一种，以免造成阅读的不方便。

4.1.3. 小括号()和函数、关键词等

小括号、关键词和函数遵循以下规则：

a．不要把小括号和关键词紧贴在一起，要用一个空格间隔；如 if(a<b)；

b．小括号和函数名间没有空格；如 test=date("ymdhis")；

c．除非必要，不要在 Return 返回语句中使用小括号。如 Return a；

4.1.4. 符号书写

在程序中=符号的书写遵循以下规则：

a．在=符号的两侧，均需留出一个空格；如 a=b、if(a==b)等；

b．在一个申明块，或者实现同样功能的一个块中，要求=号尽量上下对其，左边可以为了保持对齐使用多个空格，而右边要求空一个空格；如下例：

testa = aaa;

testaa = bbb;

testaaa = ccc;

4.1.5. if else swith for while 等书写

对于控制结构的书写遵循以下规则：

a．在 if 条件判断中，如果用到常量判断条件，将常量放在等号或不等号的左边，例如：

if(6==errorNum)，因为如果你在等式中漏了一个等号，语法检查器会为你报错，可以很快找到错误位置，这样的写法要多注意；

b．switch 结构中必须要有 default 块；

c．在 for 和 wiile 的循环使用中，要警惕 continue、break 的使用，避免产生类似 goto 的问题；

4.1.6. 类的构造函数

如果要在类里面编写构造函数，必须遵循以下规则：

a．不能在构造函数中有太多实际操作，顶多用来初始化一些值和变量；

b．不能在构造函数中因为使用操作而返回 false 或者错误，因为在声明和实例化一个对象的时候，是不能返回错误的；

4.1.7. 语句断行,每行控制在 80 个字符以内

在代码书写中，遵循以下原则：

a．尽量保证程序语句一行就是一句，而不要让一行语句太长产生折行；

b．尽量不要使一行的代码太长，一般控制在 80 个字符以内；

c．如果一行代码太长，请使用类似.=的方式断行书写；

d．对于执行数据库的 sql 语句操作，尽量不要在函数内写 sql 语句，而先用变量定义 sql 语句，然后在执行操作的函数中调用定义的变量；

例子：

```
sql = "SELECT username,password,address,age,postcode FROM test_t";
sql .= "WHERE username='aaa'";
res = mysql_query(sql);
```

4.1.8. 不要 Magic Number

一个在源代码中使用了的赤裸裸的数字是不可思议的数字，因为包括作者，在三个月内，没人它的含义。例如：

```
if(22==foo)
```

你应该用 define()来给你想表示某样东西的数值一个真正的名字，而不是采用赤裸裸的数字，例如：

```
define("STUDENT_NUM", "22"); if(STUDENT_NUM == foo)
```

4.1.9. true/false 和 0/1 判断

遵循以下规则：

a．不能使用 0/1 代替 true/false，在 Java 中，这是不相等的；

b．不要使用非零的表达式、变量或者方法直接进行 true/false 判断，而必须使用严格的完整 true/false 判断；

如：不使用 if(a)或者 if(checka())而使用 if(FALSE!=a)或者 if(FALSE!=check())

4.1.10. 避免嵌入式赋值

在程序中避免下面例子中的嵌入式赋值：

不使用这样的方式：

```
while(a!=(c=getchar()))
{
    process the character
}
```

4.1.11. 错误返回检测规则

检查所有的系统调用的错误信息，除非你要忽略错误。

为每条系统错误消息定义好系统错误文本，并记录错误 LOG。

4.2. 程序注释

Java 程序有两类注释：实现注释（implementation comments）和文档注释（document comments）。实现注释是那些在 C++中见过的，应用/*...*/和//界定的注释。文档注释（被称为"doc comments"）是 Java 独有的，并由/**...*/界定。文档注释可以经由过程 javadoc 对象转换成 HTML 文件。

实现注释用以注释代码或者实现细节。文档注释从实现（implementation-free）的角度描述代码的规范。它可以被那些手头没有源码的开辟人员读懂。

注释应被用来给出代码的总括，并供给代码自身没有供给的附加信息。注释应当仅包含与浏览和懂得程序有关的信息。例如，响应的包如何被建树或位于哪个目次下之类的信息不该包含在注释中。

在注释里，对设计决定计划中首要的或者不是显而易见的处所进行申明是可以的，但应避免供给代码中已清楚表达出来的反复信息。多余的的注释很轻易过期。凡是应避免那些代码更新就可能过期的注释。

重视：频繁的注释有时反应出代码的低质量。当你感觉被迫要加注释的时辰，推敲一下

重写代码使其更清楚。

注释不该写在用星号或其他字符画出来的大框里。注释不该包含诸如制表符和回退符之类的特别字符。

实现注释的格局（Implementation Comment Formats）

程序可以有 4 种实现注释的风格：块（block）、单行（single-line）、尾端（trailing）和行末（end-of-line）。

4.2.1. 块注释（Block Comments）

块注释凡是用于供给对文件，办法，数据布局和算法的描述。块注释被置于每个文件的开端处以及每个办法之前。它们也可以被用于其他处所，比如办法内部。在功能和办法内部的块注释应当和它们所描述的代码具有一样的缩进格局。

块注释之首应当有一个空行，用于把块注释和代码分别开来，比如：

```
/*
 * Here is ablock comment.
 */
```

块注释可以以/*-开首，如许 indent（1）就可以将之辨认为一个代码块的开端，而不会重排它。

```
/*-
  * Here is ablock comment with some very special
  * formattingthat I want indent（1） to ignore.
  *
  *     one
  *           two
  *                 three
  */
```

重视：若是你不应用 indent（1），就不必在代码中应用/*-，或为他人可能对你的代码运行 indent（1）作让步。

4.2.2. 单行注释（Single-Line Comments）

短注释可以显示在一行内，并与厥后的代码具有一样的缩进层级。若是一个注释不克不及在一行内写完，就该采取块注释（拜见"块注释"）。单行注释之前应当有一个空行。以下是一个 Java 代码中单行注释的例子：

```
if（condition）{
    /* Handle thecondition. */
    ...
}
```

4.2.3. 尾端注释（Trailing Comments）

极短的注释可以与它们所要描述的代码位于同一行，然则应当有足够的空白来分隔代码和注释。如有多个短注释呈现于大段代码中，它们应当具有雷同的缩进。

以下是一个 Java 代码中尾端注释的例子：

```
if（a==2）{
    return TRUE;                /* special case */
}else{
    return isPrime（a）;        /* works only for odda */
}
```

4.2.4. 行末注释（End-Of-Line Comments）

注释界定符"//"，可以注释掉整行或者一行中的一项目组。它一般不消于连气儿多行的注释文本；然而，它可以用来注释掉连气儿多行的代码段。以下是所有三种风格的例子：

```
if（foo>1）{
    // Do adouble-flip.
    ...
```

```
}
else{
        return false;                   // Explain why here.
}
//if（bar>1）{
//
//      // Do a triple-flip.
//      ...
//}
//else{
//      return false;
//}
```

4.3. 其他规范（建议）

4.3.1. 供给对实例以及类变量的接见把握

若没有足够来由，不要把实例或类变量声明为公有。凡是，实例变量无需显式的设置（set）和获取（gotten），凡是这作为办法调用的边沿效应（side effect）而产生。

一个具有公有实例变量的恰当例子，是类仅作为数据布局，没有行动。亦即，若你要应用一个布局（struct）而非一个类（若是 java 支撑布局的话），那么把类的实例变量声明为公有是合适的。

4.3.2. 引用类变量和类办法

避免用一个对象接见一个类的静态变量和办法。应当用类名调换。例如：

```
classMethod();                  //OK
AClass.classMethod();           //OK
anObject.classMethod();         //AVOID!
```

4.3.3. 常量（Constants）

位于 for 轮回中作为计数器值的数字常量，除了-1，0 和 1 之外，不该被直接写入代码。

4.3.4. 变量赋值（Variable Assignments）

避免在一个语句中给多个变量赋雷同的值。它很难读懂。例如：

```
fooBar.fChar=barFoo.lchar=""c"";// AVOID!
```

不要将赋值运算符用在轻易与相等关系运算符混合的处所。例如

```
if（c++=d++）{          // AVOID!（Java disallows）
    …
}
```

应当写成:

```
if（（c++=d++）!=0）{
    …
}
```

不要应用内嵌赋值运算符试图进步运行时的效力，这是编译器的工作。例如：

```
d = （a = b + c） + r;          // AVOID!
```

应当写成

```
a = b + c;
d = a + r;
```

5. 体系结构设计

5.1. 体系结构

在系统中使用浏览器-服务器（Browser/Server，B/S）体系结构，采用 B/S 的三层体系

结构，将系统的整个业务应用划分为表示层、业务逻辑层和数据访问层，这样有利于系统的开发、维护、部署和扩展。B/S 结构的基本原则是将计算机应用任务分解成多个子任务，由多台计算机分工完成，即采用“功能分布”原则。客户端完成数据处理，数据表示以及用户接口功能；服务器端完成 DBMS 的核心功能。这种客户请求服务、服务器提供服务的处理方式是一种新型的计算机应用模式。

5.2. 体系结构实现

对于这一种系统，采用如下方式进行实现：

（1）采用如下的服务端、客户端要求：

住宅维护系统客户端使用现代网页浏览器进行访问及操作，浏览器需 Internet Explorer 11、Google Chrome 62、Mozilla Firefox 57、Safari 10 以及更高的版本以获得更好的体验。

住宅维护系统服务端需运行在一个单独的服务器中，同时服务器使用 MySQL 数据库系统软件提供教学系统网站后台数据存储。

（2）采用如下的服务端、客户端配置：

服务器：以 Apache 或 Nginx 为 Web 服务器，服务器端采用 Java 语言编写，数据库采用 MySQL。

客户端：浏览网页主要采用 Internet Explorer 11、Google Chrome 62、Mozilla Firefox 57、Safari 10 以及更高的版本，通过 AJAX 技术进行网页的动态交互，同时增加服务器带宽利用率，并使用 JQuery 对客户端进行美化。

5.3. 系统运行环境

类别	标准配置	最低配置
服务器	PentiumII500以上	PentiumII500

计算机硬件	128G内存，20G硬盘， CD-ROM 16倍速以上	64G内存，20G硬盘，CD- ROM16 倍速以上
网卡	10MB/100MB自适应	10MB/100MB自适应
内存	128M 以上	128M
其他	浏览器版本：IE8 以上	浏览器版本在IE8以上

5.4. 测试环境

类别	标准配置	最低配置
服务器	PentiumII500以上	PentiumII500
计算机硬件	128G内存，20G硬盘， CD-ROM 16倍速以上	64G内存，20G硬盘，CD- ROM 16倍速以上
网卡	10MB/100MB自适应	10MB/100MB自适应
内存	128M以上	128M
其他	浏览器版本：IE8	浏览器版本在IE8以上

5.5. 运行环境

类别	标准配置	最低配置
计算机硬件	64G内存	64G内存，20G硬盘，CD- ROM 16倍以上
网卡	10MB/100MB自适应	10MB/100MB自适应
内存	128M以上	128M
其他	浏览器版本：主流浏览器	浏览器版本在IE8以上