文件修改控制页

修改记录编号	修改内容	修改人	修改日期

[附加说明]

1. “修改记录编号”的填写内容为：本次修改后的版本号＋“/”＋流水号，例如：V1.01/1。
2. 一次修改可以修改文档的多个位置，流水号为对该版本修改的流水号。当版本变时，流水号归为1。

目　录

1 命名规范

1.1 package (*)

包名全部由小写的 ASCII 字母组成，用“.”分隔。

在此项目中，所有的包均以“com.prosten.ticket”开头。

1.2 class (*)

类名应当是名词，每个内部单词的头一个字母大写。应当使你的类名简单和具有说明性。用完整的英语单词或约定俗成的简写命名类名。

【示例】public class UserManager

1.3 interface(*)

接口名应当是名词，每个内部单词的头一个字母大写。应当使你的接口名简单和具有说明性。用完整的英语单词或约定俗成的简写命名接口名。

【示例】interface TicketManagement

1.4 Class 成员属性及变量的命名 (*)

变量名全部由字母组成，头一个字母小写，以后每个内部单词的头一个字母大写。

变量名应该短而有意义。变量名的选择应该易于记忆。

一个字符的变量名应避免，除非用于临时变量。通常临时变量名的命名规则为：i，j，k，m，n用于整数；c，d，e用于字符。

【示例】private String lastName;

1.5 常量的命名(*)

Java 里的常量，是用 static final 修饰的，应该用全大写加下划线命名，并且尽量指出完整含义。

【示例】static final String SMTH_BBS="bbs.tsinghua.edu.cn";

1.6 数组的命名(*)

数组应该总是用下面的形式来命名：

```
byte[] buffer;
```

1.7 方法的参数(*)

和变量的命名规范一致，且应使用有意义的参数命名，如果可能的话，使用和要赋值的字段一样的名字。

【示例】

```
setCounter(int size){
    this.size = size;
    }
```

1.8 方法命名(*)

方法的命名应当使用动词，头一个字母小写，以后每个内部单词的头一个字母大写。

在方法名的选择上应意义明确便于记忆。

对于属性的存取方法，应使用 getXXX()和 setXXX()名称，以 isXXX()，hasXXX()来命名返回值为 boolean 类型的方法。

1.9 一般命名注意事项

用有意义的名字命名变量

首先，用完整的英语单词或约定俗成的简写命名变量。

【示例】firstName

zipCode

用复数命名 Collection 类变量。Collection 包括数组，Vector 等。命名时使用复数：

【示例】customers

classmates

2 Java 源文件样式

Java(*.java) 源文件应遵守如下的样式规则：

2.1 Class 代码布局：

版权声明

Package 和 Import 语句

Javadoc 注释或者其它文件头注释

类或接口声明

Fields 声明

空行

构造函数

空行

克隆方法

空行

其它方法（不包括 main）

空行

内部（Inner）类

空行

main()方法

2.2 版权声明

所有的源文件都应该以一个 c 风格的注释开始，以列出类名，版本信息，修改日期和版权声明。

【示例】

```
/*
 * Class name :
 *
 * Version information :
 *
 * Date :
 *
 * Copyright 2003 Prosten Technology Co.,Ltd.
 *
 */
```

其他不需要出现在 javadoc 的信息也可以包含在这里。

2.3 Package/Imports(*)

package 行要在 import 行之前，中间空一行。

将 import 的 classes 归类，按顺序罗列：

a. Java 标准类(java.*)

b. Java 扩充类(javax.*)

c. 第三方类

d. 你的应用程序的类

注意在第三方类里进行注释，说明它们的来源。如果 import 行中包含了同一个包中的多个类，则可以用 * 来处理。

【示例】

```
package com.prosten.ticket.ticketmanagement;

  import java.io.*;

  import java.util.Observable;

  import java.util.Date;

  import javax.sql.*;

 //Apache Xerces

  import org.apache.xml.*;

  import org.apache.xerces.dom.*;

  //Application classes

  import com.prosten.util.*;
```

这里 java.io.* 使用来代替 InputStream 和 OutputStream 的引入。

2.4 Javadoc 注释

【示例】/**

```
* <p>Title: 类名</p>
* <p>Description:（说明用中文）/</p>
 * @author:
* @date:（最后一次修改的提交时间）
*/
```

2.5 Class Fields

类的成员变量:

【示例】protected int[] packets;

public 的成员变量必须以生成文档（JavaDoc） 的方式进行注释（/** … */）。

proceted、private 和 package 定义的成员变量如果名字含义明确的话，可以没有注释。

Field 定义可遵从以下顺序:

a. public 常量

b. public 变量

c. protected 常量

d. protected 变量

e. package 常量

f. package 变量

g. private 常量

h. private 变量

2.6 存取方法(getter, setter)

接下来是类变量的存取的方法。

2.7 构造方法(*)

重载的构造方法应该用递增的方式写（参数多的写在后面）。

【示例】
```
public CounterSet(){
this(10);
}
public CounterSet(int size){
this.size = size;
}
```

2.8 克隆方法

如果这个类是可以被克隆的，就应实现 clone 方法:

【示例】
```
public Object clone() {
    try {
        CounterSet obj = (CounterSet)super.clone();
        obj.packets = (int[])packets.clone();
        obj.size = size;
        return obj;
```

```
        }catch(CloneNotSupportedException e) {
                throw new InternalError("Unexpected
            CloneNotSUpportedException: "
                + e.getMessage());
         }
        }
```

2.9 类方法

下面开始写类方法:

【示例】

```
/**
* Set the packet counters
* (such as when restoring from a database)
*/
protected final void setArray(int[] r1, int[] r2, int[] r3, int[] r4)
        throws IllegalArgumentException {
    if (r1.length != r2.length || r1.length != r3.length
            || r1.length != r4.length) {
        throw new IllegalArgumentException("Arrays must be he same size");
        }
    System.arraycopy(r1, 0, r3, 0, r1.length);
    System.arraycopy(r2, 0, r4, 0, r1.length);
}
```

2.10 toString 方法

每一个类都最好定义 toString 方法:

【示例】

```
public String toString() {
        String retval = "CounterSet: ";
        for (int i = 0; i < data.length(); i++) {
           retval += data.bytes.toString();
        retval += data.packets.toString();
        }
        return retval;
        }
```

2.11 main 方法 (*)

如果类中包含 main(String[]) 方法, 那么它应该写在类的底部。

3 代码编写风格

3.1 语句

3.1.1 简单语句

每一行包含至多一条语句。

3.1.2 复合语句

复合语句是指附加形如"{……}"封套结构的语句。

- 封套内的语句要比复合语句多缩进一个层次。
- 开头的括号因该在起始复合语句同一行的末尾；结尾的括号应该新起一行并和起始的复合语句保持同样缩进。
- 应当对所有诸如 if-else，while，for，try-catch 结构的控制语句都使用大括号，即使是单个语句，只要它是控制结构的一部分。

【示例】
```
if (condition) {
    statements;
} else {
    statements;
}  //if-else 语句
```

3.1.3 返回语句

有值返回的返回语句不应该使用括号，除非某些情况下为了使得返回值更加明显。

【示例】
```
return;
return myDisk.size();
return (size ? size : defaultSize);
```

3.2 位置控制

3.2.1 缩进

应当用四个空格作为缩排的单位。不要在源文件中保存 Tab 字符(!!)。以免在使用不同的源代码管理工具时 Tab 字符将因为用户设置的不同而显示为不同的宽度。

3.2.2 行的长度

避免行长超过 80 个字符，因为这样不好被大多数终端显示和工具处理。

3.2.3 折叠的行

当表达在一行放不下时，根据下面的一般原则打断它：

- 在一个逗号后打断。
- 在运算符前打断。
- 高层次的打断优于低层次的打断。
- 让新起的行与上一行同一层次表达的开头对齐。
- 如果上面的方法导致代码混乱或者代码减少了合适的页边空白，那么使用缩进 8 个空格代替。

【示例】下面的例子打断了方法调用：

```
someMethod(longExpression1, longExpression2, longExpression3,
        longExpression4, longExpression5);
```

```
var = someMethod1(longExpression1,
                  someMethod2(longExpression2,longExpression3));
```

【示例】下面两个例子打断了算术表达式。第一个较好，因为打断发生在较高层次上。

```
longName1 = longName2 * (longName3 + longName4 - longName5)
                + 4 * longname6;
```

【示例】语句的行折叠通常用 8 空格，这是因为 4 空格会使主体部分看起来困难。

```
//USE THIS INDENTATION INSTEAD
if ((condition1 && condition2)
        || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}
```

3.3 空白处理

3.3.1 空行

空行通过分开局部相关的代码部分，增加了可读性。

在下列情况下总是使用两个空行：

- 源文件的不同部分之间
- 和接口定义之间

在下列情况下总是使用一个空行：

- 在方法之间
- 方法里面的局部变量声明和它的第一个语句之间
- 在块注释（参见 4.1.1 节）或单行注释（参见 4.1.2 节）之前
- 方法里面的逻辑部分之间，以提高可读性

3.3.2 空格

应该在下列情况下使用空格：

- 关键字和后面的括号之间应该使用一个空格。例如：

```
while (true) {
    …
}
```

注意在方法名和后面的括号之间不应该使用空格。这有助于分清关键字和方法调用。

- 逗号之后应该使用一个空格。
- 除了“.”之外的所有二元操作符应该用空格和操作数分开。对于一元操作符不使用空格。

【示例】

```
a = (a + b) / (c * d);
while (d++ = s++) {
    n++;
}
printSize("size is " + foo + "\n");
```

- for 语句中的表达式应当用空格分开。

【示例】for (expr1; expr2; expr3)

- 强制类型转换应当跟随一个空格。

【示例】myMethod((byte) aNum, (Object) x);

```
myMethod((int) (cp + 5), ((int) (i + 3)) + 1);
```

3.4 声明

3.4.1 每行一个

每行只能书写一个声明，因为这有利于注释。

【示例】

```
int level; // indentation level
int size;  // size of table
```

【示例】上面的例子中使用一个空格分隔类型和标示符。另一种方法是使用 tab，如：

```
int     level;          // indentation level
int     size;           // size of table
Object  currentEntry;   // currently selected table entry
```

3.4.2 初始化

试着在局部变量声明的时候进行初始化。如果不在变量声明的时候进行初始化，唯一的理由就是它的初始值首先依赖于一些计算。

3.4.3 位置

声明只放在代码块的开始部分。（一个代码块是指由括号"{"和"}"包围的代码。）不要等到变量使用时才声明它们。唯一的例外是在 for 循环语句中。

【示例】

```
void myMethod() {
        int intl = 0;          // beginning of method block
        if (condition) {
            int int2 = 0;      // beginning of "if" block
           …
            }
        }
        for (int i = 0; i < maxLoops; i++) { … }
```

要避免局部变量的声明出现在比其更高的层次中。不要声明和内部代码块同名的变量。

4 程序编写规范

4.1 使用方法来访问实例变量和类变量(*)

如果没有很好的理由，一般不应将实例变量或类变量设为 public，将变量设为 public 的典型应用是此类代表一个“数据结构”，而不包含任何方法。

4.2 引用类变量和类方法(*)

避免使用对象引用来访问类（static） 变量或类方法，而应使用类名来访问。如：

```
classMethod(); //OK

AClass.classMethod(); //OK

anObject.classMethod(); //避免!
```

4.3 常量(*)

数字常量不应直接在编码中出现，除非是 for 循环中用于计数的 —1，0，或1。

字符串常量尽量不直接在编码中出现。

4.4 ?前的逻辑运算表达式

?前的逻辑运算表达式应以括号括起，如：

```
(x >= 0) ? x : -x;
```

4.5 变量赋值

避免在一个语句中赋给几个变量同样的值。这是难于阅读的。

【示例】`fooBar.fChar = barFoo.lchar = 'c'; // AVOID!`

不要在容易和相等操作符混淆的地方使用赋值操作。

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

...
```

```
}
```

而应该写成这样:

```
if ((c++ = d++) != 0) {
…
}
```

不要为了提高运行性能使用内嵌的赋值，这是编译器的任务。例如:

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

而应该写成:

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

4.6 特殊注释

在实现某个类的方法时若还未完成实际代码，但出于程序连接需要，可令该方法暂时返回一个“true”一类的结果，具体实现代码留待下一步实现。

但是此部分需要用/* ××× WAITING TO IMPLEMENT… ××× */标注。

4.7 例外

申明的错误应该抛出一个 RuntimeException 或者派生的例外。

顶层的 main()函数应该截获所有的例外，并且打印（或者记录在日志中）在屏幕上。

4.8 方法的输入参数

方法的输入参数为对象时，默认的前置条件为输入实例不为空（null），除非在 API 文档中另行说明。即默认情况下，方法体中不对传入实例是否为空特别地加以判断。

4.9 方法的返回值

当方法的返回值为对象时，返回值是否可能为空应当在 API 文档中说明。若返回值可能为空，相应的条件也应同时在 API 文档中说明。

5 Struts 编码规范

5.1 Action 和 ActionForm 的 class 命名

所有的 Action 和 ActionForm 类均以有意义的英文单词加 Action 或 Form 后缀。

【示例】
```
public class LoginAction extends Action

public class LoginForm extends ActionForm
```

5.2 ActionForm 变量命名

必须保证 ActionForm 和对应的数据对象中成员变量命名（包括大小写）的一致。

5.3 Action 内部结构

Action 类一般用于处理用户的请求，如果不保持一定结构将会臃肿到难以维护；Action 内的 execute 方法应只做请求再转发的功能。

【示例】

```
public class ShowingPlanAction extends Action {


public ActionForward execute(ActionMapping actionMapping,
                             ActionForm actionForm,
                             HttpServletRequest httpServletRequest,
                             HttpServletResponse httpServletResponse)
                    throws Exception {


    String Action = httpServletRequest.getParameter("action");
        if(action.equalsIgnoreCase("saveCreate")) {
            return saveCreate(actionMapping, actionForm,
                    httpServletRequest,httpServletResponse)
        }


        .....

}


private ActionForward saveCreate(ActionMapping actionMapping,
```

```
                                        ActionForm actionForm,
                                        HttpServletRequest httpServletRequest,
                                        HttpServletResponse httpServletResponse) {
        // 将 ActionForm 中的数据复制至数据对象
        // … …

        // 调用 DAO 相应方法处理数据对象
        // … …

        // 处理成功后转到指定页面。
         return actionMapping.findForward("OperationSuccess" );
    }
}
```

5.4 在 ActionForm 和数据对象之间复制数据

在 ActionForm 和数据对象之间复制数据，使用 Jakarta 的第三方包来复制数据，这样在将来的系统环境和中文乱码等发生改变时统一处理。

【示例】

```
import org.apache.commons.beanutils.PropertyUtils;

… …

// 将数据从数据对象复制至 ActionForm
```

```
PropertyUtils.copyProperties(showingPlanModel,showingPlanForm);
// 将数据从 ActionForm 复制至数据对象
PropertyUtils.copyProperties(showingPlanForm,showingPlanModel);
```

5.5 Struts 标记库的使用

在 JSP 中能够使用 Struts 标记实现尽量不要书写 java 代码，以利于 jsp 代码的阅读与维护。Struts 标记的示例可参看 StrutsDemo.rar。

6 注释

6.1 注释格式

6.1.1 javadoc 风格的注释

用/** …… */标注的注释是 java 特有的注释，能够用 javadoc 工具提取生成类的说明文档。

用来注释每个类，接口或成员方法。这些注释应该刚好出现在声明之前。并且尽可能使用文档标签来准确进行注释。

【示例】

```
/**
 * The Example class provides …
 */
public class Example { …
```

更多的细节，参见"How to Write Doc Comments for Javadoc"，包括使用文档注释标签（@return，@param，@see）：

http://java.sun.com/products/jdk/javadoc/writingdoccomments.html

有关文档注释和 javadoc，还可以参见 javadoc 主页：

http://java.sun.com/products/jdk/javadoc/

6.1.2 程序内部说明性注释

对于不需要用 javadoc 提炼，只是用来说明程序算法逻辑的注释可采取如下两种方式：

- 采用/* …… */ 作为注释标注

可单起一行，也可跟在某行比较短的代码后面

【示例】
```
if (condition) {
    /* Handle the condition. */
    …
}

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

- 采用 // 作为注释标注

不能用于连续的多行文本注释。

当需要将一节代码注释掉的时候，推荐使用//注释符，不要使用/*……*/注释方式。

【示例】

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


        //if (bar > 1) {
        //    …
        //}
        //else {
        //    return false;
        //}
```

6.2 注释内容

6.2.1 类或接口的注释

在类的开头说明类的名称；类的描述，包括类的类别、作用、是否关联具体的数据实体或文件实体；类的最后修改时间；使用文档注释标签说明类的作者。

【示例】

```
/**
 * <p>Title: ConfigReader</p>
 * <p>Description: 从页面配置文件 page.properties 取得配置信息的类</p>
 * <p>Date: 2003-11-19</p>
 * @author unascribed
 */
```

6.2.2 类方法的注释

在方法的开头说明方法的业务逻辑、算法、在必要的时候说明输入输出参数以及抛出异常。

【示例】

```
    /**
     * 从 showing 表里查询指定时间范围内演出场次的数量，并将查询结果返回
        * @param startDate 起始时间
            * @param endDate 终止时间
            * @return 演出场次数量
     * @throws Sql1Exception 当查询错误时抛出异常
     */
          public int queryShowingCounts(String startDate, String endDate)
                throws Sql1Exception{
        .....
        }
```

6.2.3 类变量的注释

对于所有的属性为 public 的类变量需要说明其含义，对于属性为 package、protect、private 的类变量，如果变量名命名清晰易于理解的话可以不用说明。

【示例】/** 操作员被访问次数 */

```
public int UserVisitedCounts = 0;
```

6.2.4 类常量的注释

对于所有的类常量需要说明其含义

【示例】/** 进入系统的缺省用户名 */

```
static final String DEFAULT_USER_NAME = “Prosten”;
```

7 编程实践问题

7.1 exit()

exit 除了在 main 中可以被调用外，其他的地方不应该调用。因为这样做不给任何代码机会来截获退出。一个类似后台服务的程序不应该因为某一个库模块决定了要退出就退出。

7.2 垃圾收集

JAVA 使用成熟的后台垃圾收集技术来代替引用计数。但是这样会导致一个问题：你必须在使用完对象的实例以后进行清场工作。比如一个 perl 的程序员可能这么写：

```
…
{
FileOutputStream fos = new FileOutputStream(projectFile);
project.save(fos, "IDE Project File");
}
…
```

除非输出流一出作用域就关闭，非引用计数的程序语言，比如 JAVA， 是不能自动完成变量的清场工作的。必须象下面一样写：

```
FileOutputStream fos = new FileOutputStream(projectFile);
project.save(fos, "IDE Project File");
fos.close();
```

7.3 final 类

绝对不要因为性能的原因将类定义为 final 的（除非程序的框架要求）。如果一个类还没有准备好被继承，最好在类文档中注明，而不要将她定义为 final 的。这是因为没有人可以保证会不会由于什么原因需要继承它。

7.4 性能

在写代码的时候，从头至尾都应该考虑性能问题。这不是说时间都应该浪费在优化代码上，而是我们时刻应该提醒自己要注意代码的效率。比如：如果没有时间来实现一个高效的算法，那么我们应该在文档中记录下来，以便在以后有空的时候再来实现她。不是所有的人都同意在写代码的时候应该优化性能这个观点，他们认为性能优化的问题应该在项目的后期再去考虑，也就是在程序的轮廓已经实现了以后。应注意：

·不必要的对象构造

·不要在循环中构造和释放对象

7.5 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。String 类将 StringBuffer 类封装了起来，（ 以花费更多时间为代价）为开发人员提供了一个安全的接口。当我们在构造字符串的时候，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。比如：如果有一个字符串必须不断地在其后添加许多字符来完成构造，那么我们应该使用 StringBuffer 对象和她的 append() 方法。如果我们用 String 对象代替 StringBuffer 对象的话，会花费许多不必要的创建和释放对象的 CPU 时间。

7.6 换行

如果需要换行的话，尽量用 println 来代替在字符串中使用"\n"。

不要这样：

```
System.out.print("Hello,world!\n");
```

要这样：

```
System.out.println("Hello,world!");
```

或者你构造一个带换行符的字符串，至少要象这样：

```
String newline = System.getProperty("line.separator");

System.out.println("Hello world" + newline);
```

8 附录：

```
/*
 * @(#)GetSystemConfig.java
 *
 * Copyright (c) 2002-2003 Prosten Technology Co.,Ltd.
 * Suite 702, Tower W3, Oriental Plaza, No.1 Chang An Ave, Dongcheng district
Beijing
 * All rights reserved.
 *
 * 本软件为保利票务系统2.0项目.
 */

package com.prosten.ticket.common;
```

```
import java.util.*;
import java.io.*;
import com.sun.xml.tree.*;
import org.w3c.dom.*;

/**
 * <p>Title: iBusiness</p>
 * <p>Description: 从页面配置文件取得配置信息的类</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: Prosten Corp. Ltd.</p>
 * @author unascribed
 * @version 1.0
 */
public class GetSystemConfig {
    //存放 jsp 页面映射信息
    public static HashMap mapJsp=new HashMap();
    private final static String JSPURL="jspPage";

    /**
     * 根据配置文件中的 jspID 返回对应的 url 串
     */
    public String getJspURL(String strJspID){
            return (String)mapJsp.get(strJspID);
```

```
    }


    /**
     * 将配置文件解析到 HashTable 中
     */
    public static void ParseJsp(String strFile ){
        try {
            FileInputStream is = new FileInputStream(strFile);
            Document doc = XmlDocument.createXmlDocument(is,false);
            int size = XmlUtils.getSize( doc , JSPURL );
            for ( int i = 0; i < size; i++ ) {
                Element row = XmlUtils.getElement( doc , JSPURL, i );
                mapJsp.put(row.getAttribute("ID"),row.getAttribute("pageurl"));
             }
         }
         catch ( Exception e ) {
            System.out.println( e );
         }
}
}
```