代码规范条款

JAVA • CSS/HTML——CodeStyle

14331262 王诗萌

目录

前言

千里之堤，溃于蚁穴。不积跬步，无以至千里。从身边的小处出发，养成良好的习惯，我们的代码规范之路，将变得平坦无比。

第 1 章：命名规范

1.1 类名以 package 名称为前缀

类名以 package 名称为前缀（不考虑大小写），这种属于比较差的命名规范，应该避免。

1.2 异常类的名称不以 Exception 为后缀

Exception 类的名称应该以 Exception 为后缀，能充分表达其意思。

1.3 局部变量和类属性重名

局部变量和类属性重名，虽然是合法的，但是如果没有 IDE 工具颜色标识，会让人误解。

1.4 长整形数值以小写'l'结尾

对于长整形数值来说，以小写'l'结尾，会被人为人为 1，所以需要更改为大写'L'。

1.5 函数名称和类名相同

函数名称和类名相同，这个会让人疑惑，是不是构造函数，命名习惯不推荐这样做。

1.6 非 boolean 返回类型的函数名称以疑问词为前缀

如果一个函数的返回值类型不为 boolean 类型，请不要使用疑问词作为前缀，这会让人看迷惑。常见的疑问词有：is, can, has, should, could, will, shall, check, contains, equals, startsWith, endsWith

1.7 非异常类以 Exception 结尾

非异常类，但是它的名字却以'Exception'结尾

1.8 函数的参数名和其覆盖的函数名称不一致

如果覆盖一个函数，确保函数的参数和原来一致，虽然这个是合法的，但是使用不同的参数名后，会让人费解。

1.9 变参的方法不允许传入原始数据类型的数组

由于 Java 的自动包装机制，当传递原始数据类型的数组给变参方法时，编译器会再将其包装为一个容量为 1 数组存放传入的数组。这可能并非预期的处理结果。

1.10 标准变量名称

下面的变量名称和类型基本都被大家公认，如果没有充足的理由，不要改变其类型和用途，否则会让人费解： ul>

- i, j, k, m, n . int
- f . float
- d . double
- b . byte
- c, ch . char
- l . long
- s, str . String

1.11 方法的参数不要超过 5 个

如果参数过多，使用对象包装（比如 Query）

1.12 太复杂的判断条件，比如有多个 and、or 的情况，加括号分类，分行

这样代码的可读性会好很多

第 2 章：代码风格

2.1 C 语言风格的数组创建

在 Java 代码中，使用 C 语言风格语法创建数组，而不是 Java 方式的。 int[] xxx 这种方式更容易被人理解：类型先行。

2.2 声明包含 javadoc 错误

声明通常包含的 javadoc 错误如下：1 没有 javadoc 2 必要的 javadoc tag 丢失 3 不合法的 javadoc tag 4 描述丢失，如参数描述，exception 描述等 5 名称不对，如参数名不匹配 6 多余的 javadoc tag，如参数已经删除，但是 javadoc 中还有

2.3 一次声明多个变量

在一个声明语句中声明多个变量，这个做法对 Java 不友好，应该一次声明一个变量。

2.4 Potential lost logger changes due to weak reference in OpenJDK

2.5 合理地进行参数校验

Public 类型的方法，在做业务处理之前，要保证传入的参数符合你的假设；参数校验的顺序要合理，校验代价小的优先做

2.6 对用户提交的内容进行 HTML 的过滤

避免 xss 漏洞

2.7 在已有的类中增加方法，如果这个类不是自己创建的，建议加上作者，时间，注释。

2.8 在已有的方法中加入条件分支或者其他业务逻辑，一定要注释

免得这个方法的原作者都看不懂了，其他人更搞不清

2.9 DO 要尽可能利用自身的数据完成业务逻辑

就是说 DO 之间尽量不要相互依赖

2.10 避免无用的代码

没有无用的 import,没有无用的成员变量,没有无用的 private 方法避免类似 import java.io.*;的引用

2.11 方法的注释中，参数的说明和返回值的说明要写详细

可以参考 StringUtil

```
    /**
     * 检查字符串是否为<code>null</code>或空字符串
<code>""</code>。
     *
     * StringUtil.isEmpty(null)      = true
     * StringUtil.isEmpty("")        = true
     * StringUtil.isEmpty(" ")       = false
     * StringUtil.isEmpty("bob")     = false
     * StringUtil.isEmpty("  bob  ") = false
     *
     *
     * @param str 要检查的字符串
     *
     * @return 如果为空, 则返回<code>true</code>
     */

    public static boolean isEmpty(String str) {
        return ((str == null) || (str.length() == 0));
    }
```

2.12 在程序的分支和复杂的逻辑处写详细的注释

代码特殊处理（比如特殊约定，类似字符串按一定规则组合或拆分，用 List 或 Map 返回不同类型对象，用事务时事务中返回的对象）的地方要注释 在一些逻辑复杂或重要（比如出价、类目属性、时间程序等）的代码中修改，建议加上 作者，时间，注释。

2.13 DO的每个属性要给出完整的注释

方便调用者使用，可以跟数据字典对应起来

第 3 章：流程控制

3.1 if.else 操作更改为三元操作符(?:)

某些情况下，if.else 操作可以更改为三元操作符(?:)，这样更精简。

3.2 多余的 if 判断

某些场合下，if 判断可以精简为一个赋值或者 return 语句。

3.3 接口调用是否进行了超时设置

远程的调用（如使用 hessian、httpclient）必须设置超时，比如连接超时 1 秒，读取数据超时 2 秒

第 4 章：异常处理

4.1 捕获一般异常

报告代码中 catch 一般异常，处于清晰、精确考虑，推荐捕获特定异常，这样更便于我们理解代码。一般异常包括：

- java.lang.Throwable
- java.lang.Exception
- java.lang.RuntimeException

4.2 finally 中包含 return 语句

在 finally 块中包含 return 语句，虽然有意为之，但是会导致调试困难。

4.3 catch 语句中的 throw 语句忽略了已经捕获的错误

在 catch 语句中，可能会再抛出一个新的异常，这个时候，需要封装已经捕获的 exception，而不是构建一个新的异常，不然异常的原信息就会丢失，给调试增加难度。

4.4 不允许捕获 IllegalMonitorStateException

IllegalMonitorStateException 通常只会在代码本身包含设计上的缺陷时出现（在没有持有对象锁的对象上调用 wait 或 notify 方法），因此不应被捕获。

4.5 捕获所有 Exception 时不应忽略对 RuntimeException 的处理

直接捕获所有 Exception 是常见的集中捕获各类异常的做法，但却最容易忽略对 RuntimeException 的判断，这可能导致非预期的 RuntimeException 被意外吞掉。因此，当捕获所有 Exception 时，请务必考虑是否需要重新抛出 RuntimeException。

4.6 try 语句嵌套

在一个 try 语句中再嵌套一个 try 语句，这样的代码让人迷惑，应该考虑 try 合并。

4.7 慎重抛出异常

慎重抛出异常,其实异常实例化也是影响性能的，不要用异常来代替业务逻辑的错误，能用程序发现的，不要通过异常来处理

4.8 DAO 和 Manager 大多直接抛出异常，AO 捕获进行翻译和记录日志

这个是一般约定

4.9 不允许捕获异常而不做任何操作

如果捕获的异常不需要重新抛出，必须记录最原始的异常。比如 Web 层提示给用户的是友好的出错(异常)说明，但是原始异常要记录

4.10 如果使用 log.warn 记录跟踪调试信息，一定要注意量的问题

如果使用 log.warn 记录跟踪调试信息，一定要注意量的问题，避免把服务器磁盘撑暴，没用了就拿掉

4.11 Log.xxxx(‘xxx’, exception)，第二个参数才是异常

Log.xxxx(‘xxx’, exception)，第二个参数才是异常，如果不指定，错误堆栈不会打印出来

4.12 在 log.debug 前使用 log.isDebugEnable

如果有连续的 log.debug 或 log.debug 的参数是字符串的拼接或通过一些计算生成出来，在 log.debug 前使用 log.isDebugEnable

4.13 系统调试过程中写的日志，在提交时要去掉

系统调试过程中写的日志，在提交时要去掉，或者用 if (log.isDebugEnabled())
{ log.debug("updateBidAuction:" + bidAuction);}这样的方式

4.14 打印异常信息时候加上关键信息

Log.xxx(“do xxx failure, id=” + id + “, status=” + status, e);

4.15 处理异常时候注意异常的发生

注意在 catch 块中的 NPE 等异常。

第 5 章：Class 度量

5.1 匿名类包含太多的方法

一个匿名类包含太多的方法，这个不便于理解，你可能需要考虑将其声明为内部类，这样逻辑更清晰。

5.2 覆盖 equals 方法且没有在其中最终调用 super.equals()的类必须同时也覆盖 hashCode 方法

hashCode 的设计要求必须满足“通过 equals 判断相等的两个对象，hashCode 必须相等”。

5.3 类继承 java.lang.Object

所有的 Java 类都会继承 Object，所有没有必要在添加 extends Object 啦

5.4 实现 Cloneable 接口的类必须覆盖 Object.clone 方法。

实现 Cloneable 接口的类必须覆盖 Object.clone 方法，这是 JDK 的约定。

5.5 Class 代码中包含其子类

当前 Class 代码中包含对其子类的引用，这样的引用让人迷惑，而且和 OO 设计冲突。

5.6 类的继承层次太深

一个类的继承层次太深，这个会给理解带来麻烦，这个时候你可能需要考虑重构。例外：继承第三方开发包的类除外（因为已经无法更改）。

5.7 类包含太多的构造函数

一个类包含太多的构造函数，可能会导致初始化错误，如果你修改一个构造函数后，没有考虑其他构造函数的情况，就可能导致相关的错误。如果一个类包含太多的构造函数，可以考虑转换为多个子类进行建模。

5.8 类包含太多的属性

一个类包含太多的属性，那么它要做的事情太多的，你知道的太多啦 :)，这个时候应该考虑分割为多个小的类，共同完成逻辑。

5.9 类包含太多的方法

一个类包含太多的函数，那么它要做的事情太多的，变得臃肿，这个时候应该考虑分割为多个小的类，共同完成逻辑。

5.10 class 没有 package 声明

class 没有 package 声明，这种 class 通常让人迷惑，有些框架处理不了这样的 class。

5.11 实现 Comparator 接口的类应该实现 Serializable 接口

实现 Comparator 接口的类当被用作构造有序集合时，只有此类背身可序列化时，构造出的集合才是可序列化的。这是一个很容易被忽略的问题，因此 Comparator 接口的类应该同时实现 Serializable 接口，以防止出现上述问题。

5.12 返回类型为 Boolean 的方法中不允许显式返回 null

返回类型为 Boolean 的方法通常可以通过自动展开机制赋值给 boolean 基本类型的变量，如果返回 null，将产生 NullPointerException。

5.13 Hessian 接口调用让系统不依赖于其他系统

对于一些不重要的 Hessian 接口调用的内容，可以通过异常和日志处理，让系统不依赖于其他系统

5.14 对于发消息通知用户的过程最好使用异步方式

对于发消息通知用户的过程最好使用异步方式，如果使用同步方式，要独立处理消息发送产生的异常，防止发消息过程的例外影响业务过程正常运行。

第 6 章：函数度量

6.1 私有方法被声明为 final 类型

私有方法意味着不能够被覆盖，所以声明为 final 类型是没有必要的。

6.2 返回类型为数组却可能返回 null 的时候，考虑返回尺寸为 0 的数组，而非 null

这样设计通常可以避免调用者对返回值为 null 的额外判断开销。

6.3 函数有多个 return 语句

一个函数有多个 return 语句，太多的 return 语句会让人迷惑，同时增加维护难度。

6.4 函数包含太多的参数

函数包含太多的参数后，会给维护和调用造成困难，应该使用一个好的方法进行重构，通常函数的参数应该在 5 个以下。

6.5 过度复杂的函数

过度复杂函数，也就是这个函数太长啦，这样的长函数不便于理解代码，所以要考虑对代码进行重构。

6.6 throws 语句中不必要的 exception

函数签名中包含了 exception 抛出声明，但是实际的实现代码中并没有这样的异常，这个会容易引发理解错误。

6.7 equals 方法中应判断传入对象是否为 null

这是 equals 方法的基本设计要求。

第 7 章：变量度量

7.1 在 for 循环中对循环因子赋值

在 for 循环中，再次给循环因子(通常为 i)赋值，如增加或降低循环因子的值，虽然是有意为之，但是这样的代码相当让人迷惑，某些情况下可能是笔误。

7.2 给函数参数赋值

将函数的参数作为变量，再次赋值，并对其进行相关操作。虽然大多数都是有意为之，但是这样的代码非常让人迷惑，可能都是笔误造成的。

7.3 不应用==或!=（而非 equals）判断 String

==只能判断引用相同，大部分情况下可能并不适用于判断字符串的内容相同，除非参与比较的两个字符串都是常量，或者通过 String.intern()方法进入了内部字符串池。

7.4 不允许用==（而非 isNaN）判断 NaN

按照 NaN 的语法定义，“没有任何数是等于 NaN 的”。因此不应用==判断 NaN，而要用 Float.isNaN 或 Double.isNaN 进行判断。

7.5 不必要的局部变量

报告各种不必要的具备变量，这些多余变量对程序理解没有任何帮助。通常有几种情形：本地变量马上返回的，本地变量赋值给另一个变量而根本没有使用，本地变量和另一变量为相同值等。

7.6 重复使用局部变量

将局部变量进行重用，给其赋一个毫不关联的值，这样的变量很让人迷惑，而且语义混乱，容易引发错误。

7.7 合理使用常量，尽量避免出现硬编码（写死的）数字、字符串

建议放入统一的参数类里面或放入本类或本方法的顶端

第 8 章：Java 5.0

8.1 for 循环使用'for each‘代替

在使用 for 语句对集合和数值操作时，可以考虑使用 Java 5.0 的'for each'代替。

8.2 indexOf()可以使用 contains()代替

在调用 String.indexOf()作为条件判断时，可以考虑使用 Java 5.0 的 String.contains()。

8.3 不必要的装箱操作

在 Java 5.0 中，装箱操作对基本类型，如 int, boolean，都是自动进行的，所以不需要再将基本类型进行对象转换。

8.4 不必要的拆箱操作

在 Java 5.0 中，拆箱操作对某些对象，如 Integer, Boolean，都是自动进行的，所以不需要再对这些对象进行拆箱操作。

第 9 章：性能

9.1 String.equals("")

不少人调用 String.equals("")用来判断字符串是否为空，这个会有一些性能问题，可以考虑用 String.length() == 0 替代。

9.2 使用 StringBuilder 而不是 StringBuffer

任何变量声明为 java.lang.StringBuffer 可以考虑使用 java.lang.StringBuilder，StringBuilder 是非线程安全的，所以效率更高一些。

9.3 String 变量的 length()==0 可以使用 isEmpty()替换

在判断一个空字符串时，调用 size()==0 可以考虑使用.isEmpty()替换。

9.4 Boolean 的构造函数

如果要实例化一个 Boolean 对象，使用 new Boolean 是不必要的，如果大量使用，会引发性能问题。代码中的"true"是一个字符串变量。

9.5 在类内部调用简单的 get 方法

如果在类的内部如果想获取属性值时，使用了 get 方法，且该 get 方法只包含返回逻辑，仅仅返回属性值，这个时候应该考虑直接访问变量，这样会有微小的性能提升。

9.6 手动拷贝数组

在进行数字内容 copy 时，可以考虑使用 System.arraycopy()。

9.7 避免在循环中使用“+”拼接字符串。

循环中使用“+”拼接字符串将在每一次循环中都把 String 转换为 StringBuffer/StringBuilder，拼接完成后又转换回 String，这样可能导致大量不必要的冗余开销。建议显式的使用 StringBuffer 或 StringBuilder 通过循环拼接字符串。

9.8 字符串到基本类型转换时不必要的临时对象

在将字符串转换为基本类型时，例如：new Integer("3").intValue()，这里会创建一个临时对象，应该考虑使用 Integer.valueOf("3")

9.9 基本类型到字符串时不必要的临时对象

将基本类型转换为字符串时，例如 new Integer(3).toString()，这里会有临时对象，应该使用：Integer.toString(3)，如果采用 3+""就更不可取啦。

9.10 使用 java.lang.reflect 包

如果你使用 java.lang.reflect 包下的类，要考虑一下，虽然很强大，但是 Java 中的反射是很慢的，这个要考量一下。

第 10 章：安全

10.1 访问系统属性

在代码中访问系统属性，然后根据这些值进行条件判断和逻辑执行，这个通常是不安全的，等于代码有依赖于系统的属性，如果有人恶意更改系统的属性，那么可能会出现可怕的结果。

10.2 调用 Runtime.exec()时使用非常量字符串

在调用 Runtime.exec()时，如果变量不是常量，那么依据动态变量构建的可执行命令可能包含非法指令，从而导致系统遭受攻击。

10.3 不安全的随机数生成规则

在使用 java.lang.Random 或 java.lang.math.Random()时，如果考虑完全不重复的可能性，java.secure.SecureRandom 是更好的选择，它提供了更安全的随机数生成算法。

10.4 不能相信客户端的校验，服务器端仍要校验

怀疑一切，不留下任何一个机会，这样你的程序才会健壮。

10.5 在单例的类中谨慎使用成员变量

在单例的类中谨慎使用成员变量（比如 DAO、Manager、Screen、Action、Control），避免线程安全问题和内存泄漏问题

10.6 更新、删除操作，一定要校验操作人是否有对操作数据的操作权限

更新、删除操作，一定要校验操作人是否有对操作数据的操作权限（比如删除店铺分类，分类 ID 是链接传递的，要校验传递的店铺分类 ID 是否是属于操作人的）

10.7 保证程序是线程安全的

比如：如确保没有在 servlet、screen、action 等类存放 request.scope 的私有变量

10.8 过度防御

避免出现过度防御，比如：web 层判断非空，ao 层判断非空，manager 判断非空，dao 又判断非空。

第 11 章：资源管理

11.1 I/O 资源打开后没有安全关闭

通常情况下，IO 资源会在 try 区块中打开，在 finally 区块中关闭，这样资源才能资源才能被有效关闭。如果资源不能有效地关闭，可能会导致资源泄漏，导致无法再次打开、内存问题等。I/O 资源主要包括一下： java.io.InputStream, java.io.OutputStream, java.io.Reader, java.io.Writer and java.io.RandomAccessFile 如果你对这些 I/O 对象进行封装，也需要正确关闭。

11.2 JDBC 资源打开后没有安全关闭

JDBC 资源应该在 try 区块中打开，finally 区块中关闭，这样不会出现异常抛出后资源没有被正常关闭，从而导致各种异常。JDBC 资源主要包括如下： java.sql.Connection, java.sql.Statement, java.sql.PreparedStatement, java.sql.CallableStatement, and java.sql.ResultSet.

11.3 Socket 打开后没有被安全关闭

Socket 资源应该在 try 区块中打开，在 finally 区块中关闭，不然在异常发生时资源会被安全关闭，从而导致各种异常。Socket 的资源主要如下：java.net.Socket, java.net.DatagramSocket, and java.net.ServerSocket.

11.4 资源的使用要及时释放

比如：不断地将用户信息 put 到一个 map，但是从来没有将用户信息从这个 map 中清除

11.5 用 ThreadLocal 考虑清空。

IC 曾经出现过一次故障，往 ThreadLocal 里放了一个字条串，没有清空。高并发大流量访问下造成了 OOM。

第 12 章：线程

12.1 不允许将 Calendar 用于类的静态成员

Calendar 是天生的多线程不安全的类，将其用于类的静态成员可能导致错误的在多线程中访问。更多信息，请参考 Sun Bug #6231579 和 #6178997。

12.2 不允许将 DateFormat 用于类的静态成员

DateFormat 是天生的多线程不安全的类，将其用于类的静态成员可能导致错误的在多线程中访问。更多信息，请参考 Sun Bug #6231579 和 #6178997。

12.3 延迟初始化的类成员应声明为 volatile

延迟初始化的类静态成员，如果没有用 synchronized 的加以保护，则必须以 volatile 修饰。这是由于编译器和 CPU 的乱序机制可能导致该成员的并发访问者看到一个尚未完全初始化完成的实例。

12.4 不允许在持有锁的时候调用 Thread.sleep()

在持有锁的时候调用 Thread.sleep()很可能导致等待该锁的其它线程被长时间的挂起，从而严重影响程序性能和延展性。

12.5 不允许在 Boolean 对象上使用 synchronized 关键字

由于 Boolean 对象通常仅以两个全局的常型实例出现，在其上使用 synchronized 关键字可能导致与其它共用该常型实例的完全不相关的代码形成互斥关系，这往往并不是程序设计者的初衷。

12.6 wait 应置于条件循环中是使用，wait 前检查所等待的条件已经满足，并避免意外唤醒的影响

在 wait 前判断等待的条件是否已满足可以避免在 wait 之前的 notify 通知被忽略。（尽管条件判断与 wait 两步也并不能看作原子操作）

第 13 章：代码成熟度

13.1 clone()方法出现在非 Cloneable 类中

clone()方法出现在非 Cloneable 类中，也就是该类没有实现 Cloneable 接口，这个通常是程序员的错误，忘记 implements Cloneable

13.2 对 BigDecimal 变量调用 equals 方法

在使用.equals()判断两个 java.math.BigDecimal 变量是否相等是，只有这两个变量的值和精度完全一致才相等，如 2.0 和 2.00 这两个 BigDecimal 值是不相等的，判断 java.math.BigDecimal 数学相等，需要使用.compareTo()方法。

13.3 Abstract method call in constructor

在构造函数中调用 abstract 方法，因为这个时候对象还没有初始化完成，如果在子类中实现这个 abtract 方法，会认为对象已经创建完毕，这个时候，可能会出一些异常。

13.4 通过对新实例访问静态变量

通过一个类的实例去访问静态变量，应该考虑使用类本身去访问。

13.5 给变量赋 null 值

给变量赋 null 值，可以触发垃圾收集，但是这样的结构通常会导致 NullPointerException，而且主动给变量赋 null 值，语义不明显。

13.6 在实例函数中给 static 变量赋值

在实例函数中给 static 变量赋值，虽然是合法的，但是很难保证安全性，通常会将实例变量转换为 static 变量。

13.7 调用 printStackTrace()

报告使用 Throwable.printStackTrace()方法，这个方法通常是临时调试使用，在线上环境中应该去除，而使用日志机制代替。

13.8 依恋情节

依恋情节是指在一个函数中反复调用另一个 class 的方法三次以上，我们应当考虑这个功能放在目录的类中是否合适，需要考虑将将部分逻辑转移到被调用类中。例外：调用父类、第三方开发包的类除外。

13.9 文件分隔符硬编码

在编码中使用斜杠(/)或者反斜杠(\)作为文件路径的分隔符，这样的硬编码可能会导致不能跨平台，应该使用 File.separatorChar

13.10 Overridable method call in constructor

This inspection reports any calls of overridable methods of the current class within a constructor. Methods are overridable if they are not declared **final**, **static** or **private**. Such calls may result in subtle bugs, as the object is not guaranteed to be initialized before the method call occurs.

13.11 Serializable 类没有包含 serialVersionUID 属性

Serializable 类通常会提供一个 **serialVersionUID** 属性，不然对类的更改可能会导致无法读取之前序列化版本。

13.12 String 相等判断应使用'equals()'，而不是' == '

字符串相等判断应该使用**".equals()"**，而不是**==**。

13.13 无用的赋值操作

下列情况说明变量赋值属于无用操作的： 1 变量赋值后没有进行任何读取操作 2 变量赋值后，在没有进行读取前，又被赋值

13.14 JDBC ResultSet 的索引为 0

在访问 **java.sql.ResultSet** 的 column 时，传入 0 访问第一列。由于历史原因，**java.sql.ResultSet** 的列从 1 开始，而不是 0，否则会导致 JDBC 错误。

13.15 使用废弃的集合类型

java.util.Vector 和 **java.util.Hashtable**，尽管 Java 是支持的，但是这两个类在 Jdk 1.2 中已经废弃，所以尽量使用新的集合类型。

13.16 使用 sun package 下的类

sun.*下的开发包在不同的 JVM 下实现不一样，不具有可移植性，建议不要使用。

13.17 equals()和 hashCode()不成对出现

equals()和 hashCode()通常都是成对出现，如果我们只覆盖其中的一个方法，可能会导致潜在错误，如向 Collection.中添加此类对象。

第 14 章：Java Collection

14.1 集合变量的类型为具体类，而不是接口

在声明集合变量时，不应该使用具体的类，而且合适的集合接口。

14.2 过度类型强制转换

在进行类型强制转换时，不要过度，适当的类型即可。如将一个对象强制转换为 `ArrayList`，但是实际上 `List` 就可以啦。过度强制转会导致 `ClassCastException` 错误，而且也会给测试带来麻烦。

第 15 章：JUnit

15.1 setUp()方法没有调用 super.setUp()方法

在 JUnit 的 setUp 方法中，没有调用 **super.setUp()**，这个可能会导致 TestCase 没有完全初始化。

15.2 tearDown()方法没有调用 super.tearDown()方法

在 JUnit 的 tearDown()方法中，没有调用 **super.tearDown()**，这个会导致 TestCase 的资源没有完全释放。

15.3 在产品代码(product source)中包含测试代码

JUnit 的 Test Case 代码不能在产品代码结构下，否则会导致将测试代码发布到产品环境。

15.4 JUnit 的测试方法没有任何断言语句

如果测试方法中没有断言语句，这样的测试方法意味这个没有完成或者是无说服力的，对测试结果呈现不利。

15.5 JUnit 断言语句中没有包含提示消息

在使用 JUnit 的 **assertXXX()**，包含错误消息参数，在调用是应该添加消息。错误消息能够帮助我们明白测试目的。

第 16 章：iBATIS

16.1 将 null 之赋给元类型变量

将 null 值赋给元类型变量，这个要在 result 中进行设置。

16.2 新增表时，需要增加 gmt_create, gmt_modified 两字段

这两个字段主要为搜索引擎和数据迁移提供服务。

16.3 程序里只允许逻辑删除数据库中的数据

也就是说程序员被剥夺了写 delete SQL 的权利

16.4 sqlmap 中如果用到了 str2varlist 函数，则要注意的是该 oracle 函数接受最长的字符串是 4000

留意不要超长了

16.5 SQL 语句中不要有太多的分支判断

SQL 语句中不要有太多的分支判断，要有一条主线，这样便与 DBA 检查（sql 中必须的条件，不要用动态生成）

16.6 数据库的查询条件要在程序里面校验过合法性后才传入 DAO 进行查询

避免出现数据格式错误或安全漏洞

16.7 一个 DAO 方法原则上只执行 1 条 SQL

一个 DAO 方法原则上只执行 1 条 SQL，但是有分页和 merge 需要的方法除外（新的要保证，老的代码尽量重构）

16.8 尽量不要使用连表查询，尽量在程序里面做排序组合

DB 是宝贵的资源。

第 17 章：Velocity

17.1 Long 型值比较大小

long 值比较大小

17.2 Form 表单中的字段名称要用下划线和小写

防止某些变态的浏览器进行大小写转换，在服务端收到奇怪的参数

17.3 是否统一使用了公用页头页脚

注意 UED 提供的页面里面的头尾部分替换成淘宝公用的头尾文件

17.4 模板文件中用到的对象，要先判断对象是否存在

对象存在后，可以减少各种 Velocity 错误日志。

17.5 要注意区别处理第一次进入 edit 页面和提交后校验失败返回 edit 页面

避免在提示出错的页面上面丢失输入数据

17.6 保持 VM 模板的简单性、组件化

前台 VM 模板最好划分细一点，尽量不要将多个 VM 合成一个 VM，里面到处都是 if.else；

17.7 Velocity 中数值只有 Integer 类型

对于 Long 类型的变量要调用 intValue()来比较大小，Velocity 中数值的加减乘除等运算都是要用 Integer；

17.8 界面代码遵循 XHTML 规范

UI 提供过来的页面，尽量不要做改动，如果需要改动，通知 UI

17.9 模板的文件名千万不要连续两个字母大写比如 OK

因为它会被解析成 XXX_Ok_XXX.vm，这是 webx 中的一个特性。

17.10 自动去掉输入框的前后空格

如果用到 form 的配置文件，可以在 group 标签设置 defaultTrimming="true" 否则可以在 java 程序里面进行 trim

17.11 注意变量是全局的

如果$obj.test()返回 null，那么$mark 将是上一次迭代的结果。

第 18 章：Cookie

18.1 替代 cookie 的方案

本地存储、服务器端 session，在使用本地存储的时候要注意浏览器的一些限制。

18.2 Cookie 在 Http 头中的格式

1.浏览器中符合条件的 Cookie 将被放到 Request HEAD 中然后被发送 Web 服务器 Cookie cookie1=value1; cookie 2=value2; cookie 3=value3 2.Web 服务器可以通过在 HEAD 里设置 Cooke 值返回的给浏览器 Set.Cookie cookie 1=value1; Domain=.taobao.net; Expires=Sat, 11.Sep.2010 05:27:14 GMT; Path=/

18.3 在淘宝 session 框架中使用 Cookie 存储方案

淘宝 session 框架允许使用 Cookie 作为 session 的保存方案,配置请参考 taobao.session.xml.vm(或者 session.xml.vm)文件。

18.4 Cookie 的命名

总体而言 Cookie 命名应遵循尽量短的原则，具体命名可以参考以下两个原则。 1.缩写原则: 缩写后的 Cookie key 以不超过 10 个字符为宜 如：会员注册时间“regtime”命名为“rg” 最近浏览过的宝贝“wwwtaobaocom_auction_data”命名为“wd” 2.防猜原则: Cookie 的 key 值的含义不能轻易被用户猜测到,一般采用和 Cookie 本意毫无关系的单词作为 key 如：“sessionID”命名为“cookie2”“userID”命名为“cookie1”

第 19 章：其他

19.1 clone()方法没有调用 super.clone()

clone()方法没有调用 super.clone()方法，这样可能会导致新 clone 的对象没有完全初始化。

19.2 质疑的奇数判断

通常会使用 `x % 2 == 1` 进行奇数判断，但是如果是负数的话，那就不奏效啦。你可以考虑 `x % 2 != 0` 或者 `x & 1 == 1`

19.3 expression.equals("literal")应考虑替换为“literal".equals(expression)

当.equals()的参数是 String 字符串时，应该考虑将 String 字符串作为.equals()的调用者，而 expression 为函数参数，这样可以预防 NullPointerException。

第 1 章：HTML 编码规范

HTML 是一种标记语言，HTML 没有任何真正的编程语言中的循环或是流程控制语句。然而，HTML 代码的格式和风格是非常重要的，因为要经常对 HTML 代码进行维护和修改，因此 HTML 代码必须有很清晰的逻辑结构和布局,增强可读性，而使其易懂和易于维护。HTML 代码本身是不区分大小写的，但是为了更好的统一代码布局，项目中 HTML 文件标记都以小写为主。

1.1HTML 标记的关闭规范

HTML 文档的正文都应在<body></body>标记中间，而<body>标记则应包含在<html>和</html>标记之间。如：

```
<html >
<body>正文</body>
</html>
```

不同类的标记不能交叉编码：

eg: 内容

正确编码应为： 内容

开始和关闭标记放在一行的标记有:

eg:

和

<u>和</u>

和

各种标题标记，如<h1>…</h1>等

<a>和

1.2HTML 文件头基本标记

```
①<!DOCTYPE html PUBLIC ".//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1.transitional.dtd">
②<html xmlns="http://www.w3.org/1999/xhtml">
③<meta http.equiv="Content.Type" content="text/html; charset=utf.8" />
<meta http.equiv="Content.Script.Type" content="text/javascript">
<meta http.equiv="Content.Style.Type" content="text/css">
④<meta name="Robots" content="all">
⑤<meta name="keywords" content="关键词（逗号分隔）">
⑥<meta name="description" content="描述词（逗号分隔）">
<head>
⑦<link href="样式文件名.css"  rel="stylesheet" type="text/css" />
⑧<script language="javascript"  type="text/javascript" src="JS 文件名.js"></script>
<title>页面标题名</title>
       </head>
```

说明: ①和② 是 html 网页基本的标准协义,包含文件中顶部可以不用此标签。

③我们的中文环境中用 utf.8 编码,一般通常是用 GB2312 编码的，项目中用 utf.8 是为了防止编码错误显示和浏览时乱码的现像。新建文件时文件通常是 ANSI 或其它格式的,所以编码时也按照该格式的编码,容易导致浏览乱码。这点要注意检查，可以用记事本将文件另存时,选定 utf.8 格式保存文件.

④是否允许网页被其它服务器搜索到内容，all 为允许, none 为不允许.该项为可选的，不是非必要的。

⑤和⑥是方便爬虫搜索时获取关键词,取决于④状态值是 all 的情况下。该项为可选的。

⑦CSS 样式引用格式

⑧JS 引用格式

1.2HTML 正文代码标记元素

1.1 input 标记的属性值规范

对于标记中的属性值，必须使用双引号或单引号包围，这样的话不易出错。

eg: <input type="text" value="你的值" length="20" >

或者

eg: <input type=’text’ value=’你的值’ length=’20’ >

1.2 其它标签的引用,凡是属性值一律加双引号或单引号包围。

比较常用的标签有 <img src= ’ ’ >,链接,<table border =’ ’ >等等。

1.3 重点说明一下 img 图片标签，该标签必须加载 alt="图片描述文字"。以便在没有显示图片时，显示文字说明。

eg:

1.3HTML 标记的缩进规范

HTML 标记的缩进三点规范：

① 最高一级的父标记采用左对齐顶格方式书写。

② 下一级标记采用左对齐后，缩进 2 个空格的方式书写。再下一级则以此类推。

③ 同一级标记的首字符上下应对齐。承接来自不同货主的运单。

eg: 下面是两表格嵌套实例

```
<p>
<table>
    <tr>
        <td> … … … </td>
        <td> … … … </td>
    </tr>
    <tr>
        <td>
            <table>
                    <tr>
                            <td>  ……  </td>
                    <tr>
            </table>
        </td>
```

```
        <td> … … </td>
    </tr>
</table>
</p>
```

第 2 章：HTML 文件引入 CSS 样式代码和 Javascript 代码规范

2.1 引入 css 样式代码规范

在 HTML 文件中引入 css 代码，应该遵循的格式如下：

```
<style type="text/css">
CSS 样式示代码
</style>
```

开头：<style type="text/css">

中间：　　CSS 样式示代码

结尾：</style>

eg:示例如下

```
<!DOCTYPE html PUBLIC ".//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1.transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http.equiv="Content.Type" content="text/html; charset=utf.8" />
<title>在线系统—标题</title>
    <style type="text/css">
      .csstest{
                width:180px;
                hight:20px;
                color:#ff0;
}
</style>
</head>
<body>
      正文部分
      <div class=’ csstest’ >CSS</div>
</body>
</html>
```

2.2 引入 Javascript 代码规范

在 HTML 文件中引入 javascript 代码，应该遵循的格式如下：

```
<script type="text/javascript">
<!—
```

javascript 代码
//..>
</script>
开头：<script type="text/javascript">
中间：　　<!—
javascript 样式示代码
//..>
结尾：</script>
eg:示例如下
<!DOCTYPE html PUBLIC ".//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1.transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http.equiv="Content.Type" content="text/html; charset=utf.8" />
<title>在线系统—标题</title>
  <script type="text/javascript">
<!..
var a=’1234567’;
alert(a);
//..>
</script>
</head>
<body>
　　正文部分
</body>
</html>

第 3 章：HTML 注释标签<!..和..>

在任何代码中都应该有做注释的好习惯，在一个复杂的 HTML 代码中，友好的注释是非常有用的，特别是在有很多嵌套的表格中。HTML 中使用<!—和..>来做注释。
eg：<!.. 描述内容，不会被执行 ..>

第 4 章：CSS 编码规范

4.1 CSS 编码要求

所有的代码小写
属性的值一定要用双引号("")括起来，且一定要有值
每个标签都要有开始和结束，且要有正确的层次
空元素要有结束的 tag 或于开始的 tag 后加上"/"
　　表现与结构完全分离,代码中不涉及任何的表现元素,如 style、font、bgColor 等
给每一个表格和表单加上一个唯一的、结构标记 id

给重要的区块加上注释用"/* 描述内容 */"
所有的标签必须进行合理的嵌套
根元素前必须有元素，宣告使用那一种 DTD

4.2 CSS 样式表规范

id 和 class 命名采用该版块的英文单词或组合命名，每个单词都是小写，用下划线连接起来，如:content_main（主要内容区域)
CSS 样式表各区块用注释说明
使用英文命名原则
尽量不缩写，除非一看就明白的单词

4.3 CSS 命名规范

DIV	CSS 名称	说明
	网站公用相关	
Container div	#wrapper	容器
Header or banner div	#header	页头部分
Main or global navigation div	#main_nav	主导航
Menu	#menu	菜单
Sub Menu	#sub_menu	子菜单
Left or right side columns	#sidebar	左边栏或右边栏
Main div	#main	页面主体
Content div	#content	内容部分
The main content area	#content_main	主要内容区域
Block div	#block	一块区域
Footer div	#footer	页脚部分
Tag	#tag	标签
Message	#msg #message	提示信息
Tips	#tips	小技巧
Vote	#vote	投票
Friend Link	#friendlink	友情连接
Title	#title	标题
Summary	#summary	摘要
Sub.navigation list	#sub_nav	二级导航
Search input	#search_input	搜索输入框
Search output	#search_output	搜索输出和搜索结果相似
Search	#search	搜索
Search results	#search_results	搜索结果

Copyright information	#copyright	版权信息
brand	#branding	商标
branding.logo	#logo	LOGO
Site information	#siteinfo	网站信息
Copyright information etc.	#siteinfo_legal	法律声明
Designer or other credits	#siteinfo_credits	信誉
Join us	#joinus	加入我们
Partnership opportunities	#partner	合作伙伴
Services	#service	服务
Regsiter	#regsiter	注册
Status	#status	状态
电子贸易相关		
Products	.products	产品
Products prices	.products_prices	产品价格
Products description	.products_description	产品描述
Products review	.products_review	产品评论
Editor's review	.editor_review	编辑评论
New release	.news_release	最新产品
Publisher	.publisher	生产商
Screen shot	.screenshot	缩略图
FAQ	.faqs	常见问题
Keyword	.keyword	关键词
Blog	.blog	博客
Forum	.forum	论坛

4.4 样式文件命名

主要的 master.css
布局，版面 layout.css
专栏 columns.css
文字 font.css
打印样式 print.css
主题 themes.css
提交 submit.css
文本框 textbox.css
统计 count.css

4.5 样式文件布局

div：主要用于布局，分割页面的结构
ul/ol：用于无序/有序列表
span：没有特殊的意义，可以用作排版的辅助，然后在 css 中定义 span
h1.h6：标题
h1.h6：　根据重要性依次递减
h1：最重要的标题
label：为了使你的表单更有亲和力而且还能辅助表单排版的好东西
fieldset　&　legend：fildset 套在表单外，legend 用于描述表单内容。
dl,dt,dd：当页面中出现第一行为类似标题/简述，然后下面为详细描述的内容时应该使用该标签

第 5 章：CSS 常规书写规范及方法

5.1 文件调用方法:

Css 文件调用方法分为页面内嵌法和外部调用
页面内嵌法调用：
<style type=”text/css”>
<!- body { background : white ; color : black ; } ->
</style>
外部调用: 将样式表写在一个独立的.css 文件中，然后在页面 head 区用类似以下代码调用。如下：
<link rel=”stylesheet” rev=”stylesheet” href=”文件名.css” type=”text/css” media=”all” />

5.2 CSS 结构化书写

5.2.1 派生选择器：

可以使用派生选择器给一个元素里的子元素定义样式，在简化命名的同时也使结构更加的清晰化。
如：mainMenu ul li {background:url(images/bg.gif;)}

5.2.2 辅助图片用背影图处理：

这里的”辅助图片”是指那些不是作为页面要表达的内容的一部分，而仅仅用于修饰间隔、提醒的图片。将其做背影图处理，可以在不改动页面的情况下通过 CSS 样式来进行改动。
如：#logo {background:url(images/logo.jpg) #FEFEFE no.repeat right bottom;}

5.2.3 结构与样式分离：

在页面里只写入文档的结构，而将样式写于 css 文件中，通过外部调用 CSS 样式表来实现结构与样式的分离。

5.2.4 文档的结构化书写

页面 CSS 文档都应采用结构化的书写方式，逻辑清晰易于阅读。
如：

```
<div id=”mainMenu”>
     <ul>
         <li><a href=”#” >首页</a></li>
         <li><a href=”#” >介绍</a></li>
         <li><a href=”#” >服务</a></li>
     </ul>
</div>
/*=====主导航=====*/
#mainMenu {
    width:100%;
    height:30px;
    background:url(images/mainMenu_bg.jpg) repeat.x;
}
#mainMenu ul li {
    float:left;
    line.height:30px;
    margin.right:1px;
    cursor:pointer;
    }
/*=====主导航结束=====*/
```

5.3 HACK CSS 书写规范

因为不同浏览器对 W3C 标准的支持不一样，各个浏览器对于页面的解释呈视也不尽相同，比如 IE 在很多情况下就与 FF 存在 3px 的差距，对于这些差异性，就需要利用 css 的 hack 来进行调整，当然在没有必要的情况下，最好不要写 hack 来进行调整，避免因为 hack 而导致页面出现问题。

5.3.1 IE6、IE7、Firefox 之间的兼容写法

写法一：

IE 都能识别*;标准浏览器(如 FF)不能识别*;
IE6 能识别*，但不能识别 !important,
IE7 能识别*，也能识别!important;
FF 不能识别*，但能识别!important;
根据上述表达，同一类/ID 下的 CSS hack 可写为：

```
.searchInput {
background.color:#333;/*三者皆可*/
*background.color:#666  !important; /*仅 IE7*/
*background.color:#999; /*仅 IE6 及 IE6 以下*/
```

```
}
```

一般三者的书写顺序为：FF、IE7、IE6.

写法二：

IE6 可识别“_”，而 IE7 及 FF 皆不能识别，所以当只针对 IE6 与 IE7 及 FF 之间的区别时，可这样书写：

```
.searchInput {
background.color:#333;/*通用*/
_background.color:#666;/*仅 IE6 可识别*/
}
```

写法三：

*+html 与 *html 是 IE 特有的标签，Firefox 暂不支持。

```
.searchInput {background.color:#333;}
*html .searchInput {background.color:#666;}/*仅 IE6*/
*+html .searchInput {background.color:#555;}/*仅 IE7*/
```

5.3.2 屏蔽 IE 浏览器

select 是选择符，根据情况更换。第二句是 MAC 上 safari 浏览器独有的。

```
*:lang(zh) select {font:12px !important;} /*FF 的专用*/
select:empty {font:12px !important;} /*safari 可见*/
```

IE6 可识别：这里主要是通过 CSS 注释分开一个属性与值，注释在冒号前。

```
select { display /*IE6 不识别*/:none;}
```

IE 的 if 条件 hack 写法：

所有的 IE 可识别：

```
<!-[if IE]> Only IE <![end if]->
```

只有 IE5.0 可以识别:

```
<!-[if IE 5.0]> Only IE 5.0 <![end if]->
```

IE5.0 包换 IE5.5 都可以识别:

```
<!-[if gt IE 5.0]> Only IE 5.0+ <![end if]->
```

仅 IE6 可识别:

```
<!-[if lt IE 6]> Only IE 6. <![end if]->
```

IE6 以及 IE6 以下的 IE5.x 都可识别:

```
<!-[if gte IE 6]> Only IE 6/+ <![end if]->
```

仅 IE7 可识别:

```
<!-[if lte IE 7]> Only IE 7/. <![end if]->
```

5.3.3 清除浮动

在 Firefox 中，当子级都为浮动时，那么父级的高度就无法完全的包住整个子级，那么这时用这个清除浮动的 HACK 来对父级做一次定义，那么就可以解决这个问题。

```
select:after {
    content:”.”;
    display:block;
    height:0;
```

```
    clear:both;
    visibility:hidden;
}
```

5.3.4 鼠标手势

在 XHTML 标准中，hand 只被 IE 识别，当需要将鼠标手势转换为“手形”时，则将“hand”换为“pointer”，即“cursor:pointer;”。

第 6 章：CSS 性代码缩写

6.1 不同类有相同属性及属性值的缩写

对于两个不同的类，但是其中有部分相同甚至是全部相同的属性及属性值时，应对其加以合并缩写，特别是当有多个不同的类而有相同的属性及属性值时，合并缩写可以减少代码量并易于控制。
如：

```
#mainMenu {
    background:url( ‘../images/bg.gif’ );
    border:1px solid #333;
    width:100%;
    height:30px;
    overflow:hidden;
    }
    #subMenu {
    background:url( ‘../images/bg.gif’ );
    border:1px solid #333;
    width:100%;
    height:20px;
    overflow:hidden;
    }
```

两个不同类的属性值有重复之处，刚可以缩写为：

```
#mainMenu,#subMenu {
    background:url( ‘../images/bg.gif’ );
    border:1px solid #333;
    width:100%;
    overflow:hidden;
    }
    #mainMenu {height:30px;}
    #subMenu {height:20px;}
```

6.2 同一属性的缩写

同一属性根据它的属性值也可以进行简写。

如：

```
  .search {
    background.color:#333;
    background.image:url( '../images/icon.gif' );
    background.repeat: no.repeat;
    background.position:50% 50%;
    }
    .search {
    background:#333 url( '../images/icon.gif' ) no.repeat 50% 50%;
    }
```

6.3 内外侧边框的缩写

在 CSS 中关于内外侧边框的距离是按照上、右、下、左的顺序来排列的，当这四个属性值不同时也可直接缩写。如：

```
.btn {
          margin.top:10px;
       margin.right:8px;
       margin.bottom:12px;
      margin.left:5px;
      padding.top:10px;
padding.right:8px;
  padding.bottom:12px;
  padding.left:8px;
    }
```

则可缩写为：

```
.btn {
     Margin:10px 8px 12px 5px;
      Padding:10px 8px 12px 5px;
  }
```

而如果当上边与下边、左边与右边的边框属性值相同时，则属性值可以直接缩写为两个。

如：

```
.btn {
     margin.top:10px;
     margin.right:5px;
     margin.bottom:10px;
          margin.left:5px;
    }
```

缩写为：

```
.btn {margin:10px 5px;}
```

而当上下左右四个边框的属性值都相同时，则可以直接缩写成一个，如：

```
.btn {
                margin.top:10px;
           margin.right:10px;
```

```
            margin.bottom:10px;
            margin.left:10px;
        }
```
缩写为： .btn{margin:10px;}

6.4 颜色值的缩写

当 RGB 三个颜色值数值相同时，可缩写颜色值代码。如：

```
.menu { color:#ff3333;}
```
可缩写为:
```
.menu {color:#f33;}
```

第 7 章：CSS 注释书规范

7.1 行间注释

直接写于属性值后面，如：

```
.search{
            border:1px solid #fff;/*定义搜索输入框边框*/
        background:url(../images/icon.gif) no.report #333;/*定义搜索框的背景*/
    }
```

7.2 整段注释

分别在开始及结束地方加入注释，如：

```
/*=====搜索条=====*/
    .search {
        border:1px solid #fff;
        background:url(../images/icon.gif) no.repeat #333;
    }
    /*=====搜索条结束=====*/
```