链滴

JAVA 开发者常犯的 10 大错误【译】

作者：2457081614

原文链接：https://ld246.com/article/1576823062955

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20190110.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h3 id="1-将-Array-转换成-ArrayList">1.将 Array 转换成 ArrayList</h3>
通常我们将 Array 转成 ArrayList 都是使用下面的语法，
<pre><code class="highlight-chroma"><span class="highlight
cl">List<String> list = Arrays.asList(arr);
</code></pre>
Arrays.asList()方法将返回 ArrayList，它是 Arrays 中的一个私有静态类而不是 java.util.ArrayList
类。 <code>java.util.Arrays.ArrayList</code> 类具有 set ()、 get ()和 contains ()方法，但没有
何用于添加元素的方法，因此其大小是固定的。 要创建一个真正的 ArrayList，你应该这样做:
<pre><code class="highlight-chroma"><span class="highlight
cl">ArrayList<String> arrayList = new ArrayList<String>(Arrays.asList(arr));
</code></pre>
<hr>
<h3 id="2-检查数组是否包含值">2.检查数组是否包含值</h3>
常见错误做法：
<pre><code class="highlight-chroma"><span class="highlight
cl">Set<String> set = new HashSet<String>(Arrays.asList(arr));
return set.contains
targetValue);
</code></pre>
上面这个代码可以达到我们的目的，但是不需要转换 将列表转换为集合需要额外的时间。 可以
化成:
<pre><code class="highlight-chroma"><span class="highlight
cl">Arrays.asList(arr).contains(targetValue);
</code></pre>
<hr>
<h3 id="3-从循环列表中删除某个元素">3.从循环列表中删除某个元素</h3>
错误做法：
<pre><code class="highlight-chroma"><span class="highlight
cl">ArrayList<String> list = new ArrayList<String>(Arrays.asList("a", "b", "c", "d"));
for (int i = 0; i <
ist.size(); i++) {
    list.remove(i);
}
System.out.println
list);
</code></pre>
输出结果：
<code>[b, d]</code>
这种方法有一个严重的问题。删除元素时，列表的大小会缩小，索引也会发生变化。因此，如果您希
通过使用索引来删除循环中的多个元素，那将无法正常工作。
您可能知道使用迭代器是在循环中删除元素的正确方法，而且您知道 Java 中的 foreach 循环工作起
像迭代器，但实际上并非如此。代码如下。
<pre><code class="highlight-chroma"><span class="highlight
cl">ArrayList<String> list = new ArrayList<String>(Arrays.asList("a", "b", "c", "d"));

for (String s : list) {
    if (s.equals("a"))
        list.remove(s)

}
</code></pre>
它将抛出 <a href="https://ld246.com/forward?goto=https%3A%2F%2Fwww.programcree

.com%2F2014%2F01%2Fjava-util-concurrentmodificationexception%2F" target="_blank" rel=
nofollow ugc">ConcurrentModificationException。
使用迭代器进行删除能够避免这个问题，代码如下
<pre><code class="highlight-chroma"><span class="highlight
cl">ArrayList<String> list = new ArrayList<String>(Arrays.asList("a", "b", "c", "d"));
Iterator<String
gt; iter = list.iterator();
while (iter.hasNext
)) {
    String s = iter.n
xt();

    if (s.equals("a"))

        iter.remove();
    }
}
</code></pre>
<h2 id="-必须在之前调用-"><code>.next()</code> 必须在 <code>.remove()</code> 之前调
。</h2>
<h3 id="4-Hashtable-vs-HashMap">4.Hashtable vs HashMap</h3>
map 预览

HashMap：底层由哈希表实现，对键或值没有排序。
TreeMap：基于红黑树实现，并按键排序。
Hashtable：与 HashMap 相反，Hashtable 是同步的。 它有一个同步的开销。
LinkedHashMap：保留插入顺序，有序。

注意项：
如果 HashMap 的键是一个自定义对象，那么需要遵循 equals ()和 hashCode ()契约。</s
rong>
由于 TreeMaps 是按键排序的，因此 key 的对象必须能够相互比较，这就是它必须实现 C
mparable 接口的原因
###5.使用原始类型(Object)的集合
<pre><code class="highlight-chroma"><span class="highlight
cl">public static void add(List list, Object o){
    list.add(o);
}
public static void
main(String[] args){
    List<String&g
; list = new ArrayList<String>();
    add(list, 10);
    String s = list.g
t(0);
}
</code></pre>
上面这段会出现如下报错信息：
<pre><code class="highlight-chroma"><span class="highlight
cl">Exception in thread "main" java.lang.ClassCastException: java.lang.Integer cannot be cast
o java.lang.String

    at ...
</code></pre>
<h2 id="使用原始类型集合是危险的-因为原始类型集合跳过了泛型类型检查-并且不安全-">使用原类型集合是危险的，因为原始类型集合跳过了泛型类型检查，并且不安全。</h2>
###6. 访问级别
开发人员经常使用 public 作为类字段。 通过直接引用字段值很方便，但这是一个非常糟糕的设计。 确的做法是尽可能低为成员提供低的访问级别。
下表总结了成员的不同修饰符的访问级别，访问级别决定字段和方法的可访问性。 它有 4 个级别: pub ic、 protected、 package-private (没有显式修饰符)或 private。

<h3 id="7-ArrayList-vs--LinkedList">7.ArrayList vs. LinkedList</h3>
当开发人员不知道 ArrayList 和 LinkedList 之间的区别时，他们通常使用 ArrayList，因为它看来很熟悉。 然而，它们之间存在着巨大的性能差异。
集合预览图如下。

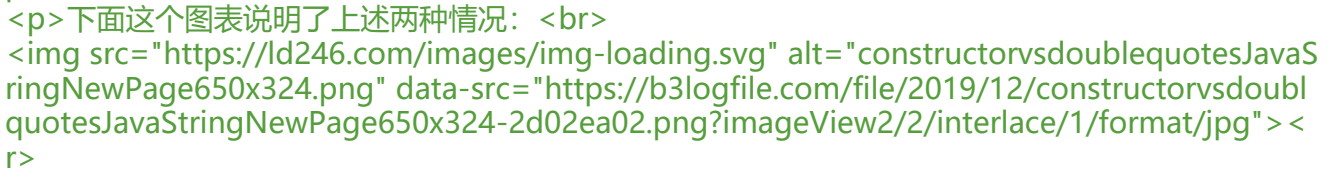
/interlace/1/format/jpg">
Arraylist 由一个可调整大小的数组实现。 随着更多的元素被添加到 ArrayList 中，它的大小会动态增。 它的元素可以通过 get 和 set 方法直接访问，因为 ArrayList 实际上是一个数组。
Linkedlist 实现为双链表。 它在添加和删除方面的性能优于数组列表，但在 get 和 set 方法上性能较。
时间复杂度比较如下：

imageView2/2/interlace/1/format/jpg">
性能比较：

lace/1/format/jpg">
他们的表现有明显的差异。 Linkedlist 在添加和删除方面更快，但在 get 方面更慢。 根据复杂度表测试结果，我们可以确定何时使用 ArrayList 或 LinkedList。 简而言之，如果下列情况下 LinkedList 应该是首选的：
* 元素没有大量的随机存取
* 有大量的添加 / 删除操作
<hr>
<h3 id="8-可变对象VS不可变对对象">8.可变对象 VS 不可变对对象</h3>
不可变对象有许多优点，如简单性、安全性等。 但是，对于每个不同的值，它需要一个单独的象，而且对象太多可能会导致垃圾收集的高成本。 在可变和不可变之间进行选择时，应该有一个平衡
通常，使用可变对象是为了避免产生过多的中间对象。 一个典型的例子是连接大量的字符串。 如果用不可变字符串，则会生成许多可立即进行垃圾收集的对象。 这会浪费 CPU 的时间和精力，而使用变对象是正确的解决方案（<code>StringBuilder</code>）
<hr>
###9.父类和子类构造器

edfordefaultconstructor.png" data-src="
uctorisundefinedfordefaultconstructor-7abcade6.png?imageView2/2/interlace/1/format/jpg"

由于未定义默认父类构造函数，因此发生此编译错误。 在 Java 中，如果类没有定义构造函数，编译将默认为类插入默认的无参数构造函数。 如果在 Super 类中定义了一个构造函数，在这种情况下，S per (String s)，编译器不会插入默认的无参数构造函数。 这就是上面提到的父类的情况。
<hr>

###10.使用""还是构造器初始化 String
<pre><code class="highlight-chroma">//1. 使用双引号
String x = "abc";
//2. 构造器方式
String y = new String("abc");
</code></pre>
下面我们分析一下他们之间的区别：
例子 1
<pre><code class="highlight-chroma">String a = "abcd";
String b = "abcd";
System.out.println(a == b); // True
System.out.println(a.equals(b)); // True
</code></pre>
a==b 为 true，因为 a 和 b 在方法区域中引用了相同的字符串。 内存引用是相同的。
当多次创建同一字符串时，每个不同字符串值只存储一个副本。 这叫做“字符串主流”。 Java 中的有编译时常量字符串都会自动实现。
例子 2
<pre><code class="highlight-chroma">String c = new String("abcd");
String d = new String("abcd");
System.out.println(c == d); // False
System.out.println(c.equals(d)); // True
</code></pre>
c == d 是 false，因为 c 和 d 引用堆中的两个不同对象。 不同的对象总是有不同的内存引用。
下面这个图表说明了上述两种情况：



原文链接