链滴

【双语】不为人知的集合异常检查工具:Collections.checkedCollection()

作者：Polly

原文链接：https://ld246.com/article/1517584869839

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

英文原文链接：https://www.javaspecialists.eu/archive/Issue251.htm
摘要: Java 在 5.0 的时候已经具备了检查集合中的元素类型是否正确的能，但是只有很少一部分程序员知道这个功能，这个功能在代码调试阶段起到了非常巨大的作用，它可在早期就把异常抛出。
<blockquote>
Dinosaurs roamed the earth. Fred Flintstone wrote Applets with JBuilder. A shaft of lightng split the dark sky and with a flash <> appeared on his amber screen. Fred scratched hs head. _"What on flat earth was List<String>?"
</blockquote>
在那古老的恐龙时代，原始人弗莱德正在用 Jbuilder 写一个 Applets 程序,一道闪电劈开了昏暗天空，<> 在弗莱德那琥珀显示器上不停的闪烁。弗莱德抓了抓脑袋说道： “List<String>是个什么鬼?”
<blockquote>
It was the advent of generics.
</blockquote>
这就是泛型的降临
<blockquote>
Programmers do not like change. Or rather, we do not like change that will break something that was working perfectly well before. Java 1.4 introduced the <code>**assert**</code> eyword. It broke a bunch of our classes. Java 1.5 added <code>**enum**</code> to the relatvely short list of reserved words, a popular name for Enumeration instances. And that was the ast time a new keyword was added to the Java Programming Language. Sun Microsystems' egineers knew their audience. They also knew that for their generics to be accepted, old code hould preferably still compile without any changes necessary.
</blockquote>
程序员不喜欢变化，不如说我们不喜欢去接受一些新的改变我们编程方式的东西，因为我们之前编程方式已经做的很熟练和完美了。Java1.4 加入了“断言” assert 关键字，改变了我们大量的 java 类，Java1.5 添加了枚举 enum （Enumeration instanes）这个流行的关键字，这也是最后一次向 Java 中添加关键字了。Sun 公司(java 的创始公司已被 Orcle 收购)的工程师了解他们的用户，并很肯定他们的用户也能接受泛型，因为使用泛型你不需要修改前的任何代码。
<blockquote>
Generics were designed so we could ignore them if we wanted to. The javac compiler wold issue a faint sigh, but would still compile everything as before. How did they do it? Tpe erasure was the magic ingredient. When they compiled the class ArrayList, the genec type parameter was erased and replaced with Object. Even the <code>E[]</code> de> was erased to <code>Object[]</code>. In Java 6, they changed the element array in ArryList to the more honest <code>Object[]</code>.
</blockquote>
泛型虽然被加入到了 Java 体系总但我们完全可以不去使用它，编译器只是会有一个警告标志，不影响程序的执行。他们怎么做到的？这就是类型擦除（Java 泛型的内部机制有兴趣的朋友可以百度一下），当我们编译 ArrayList 这个类时,泛型类型是被擦除掉的，被 Object 对象的祖宗）这个类型取代，甚至异常占位符 E[]也被替换成了 Objec[] (百度一下泛型占位符的基本概念),在 Java 1.6 版本,他们把 ArrayList 里装载数据的类型改变成了 Object[]。
<pre><code class="highlight-chroma">1. private transient Object[] elementData;
</code></pre>
有兴趣的可以去百度一下这一行代码，深入的理解一下 ArrayList 这个使用非常频繁的集合。<b>
(这部分看不明白不影响后面的阅读)
<blockquote>

By not distinguishing at runtime between ArrayList<String> and ArrayList<Intege
>, we allowed Java programmers to still shoot themselves in the foot like so:
</blockquote>
如果不去关心 ArrayList<String> 和 ArrayList<Integer> 会出现的问题，我们依然
许程序员去拿石头砸自己的脚，如下面的代码：
<pre><code class="highlight-chroma"><span class="highlight
cl">import java.util.*;
public class Foot
hootJava5{
  public static vo
d main(String... args) {
    List<String&
t; names = new ArrayList<>();
    Collections.ad
All(names, "John", "Anton", "Heinz");
    List huh = nam
s;
    List<Integer
gt; numbers = huh;
    numbers.add(4
);
  }
}
</code></pre>
<blockquote>
Sure, javac would emit a warning, but at runtime everything would appear to work. It was
only when we retrieved elements from <code>ArrayList</code> that a cast to String was inse
ted into the client code and then a <code>ClassCastException</code> would jump in our fa
es. This is an example of an exception that is thrown late. A while after the incor
ect object has been inserted into the <code>ArrayList</code>, we discover that it wasn't a St
ing after all, thus if we add the following we see the problem:
</blockquote>
这段代码并不会报错，一切运行正常，但是当我们从集合中取出数据并加以操作的时候，异常就
跳到我们脸上，这种异常是一种_后期异常（暂时这么翻译，在本例意思是异常并没有出现在不同类
数据插入时，而是在使用时才出现）_，下面的例子中解释了这一点：
<pre><code class="highlight-chroma"><span class="highlight
cl">import java.util.*;
import static java
util.stream.Collectors.*;
public class Foot
hootJava8 {
  public static vo
d main(String... args) {
    List<String&g
; names = new ArrayList<String>();
    Collections.add
ll(names, "John", "Anton", "Heinz");
    List huh = nam
s;
    List<Integer
gt; numbers = huh;
    //应该出现异常
位置,因为插入了非String类型的数据，但这一行并没有抛出异常，而是在下一行抛出了
    numbers.add(4

);
    //stream:jdk1.
新特性，可百度一下StreamAPI,下面这行代码意思是将集合中的所有元素连接在一起，以+号分隔开
    //这一行抛出了
型转换异常
    System.out.print
n(names.stream().collect(joining("+")));
    }
}
</code></pre>
(StreamAPI 相关博文:<a href="https://ld246.com/forward?goto=http%3A%2F%2Fblog.cs
n.net%2Fu010425776%2Farticle%2Fdetails%2F52346644" target="_blank" rel="nofollow ugc
>http://blog.csdn.net/u010425776/article/details/52346644)
<blockquote>
esults in a rather grumpy:
</blockquote>
结果让人很暴躁
<pre><code class="highlight-chroma"><span class="highlight
cl">ClassCastException: Integer cannot be cast to CharSequence
    at ReduceOps$
ReducingSink.accept()
    at ArrayList$Arr
yListSpliterator.forEachRemaining()
    at AbstractPipel
ne.copyInto()
    at AbstractPipel
ne.wrapAndCopyInto()
    at ReduceOps$
educeOp.evaluateSequential()
    at AbstractPipel
ne.evaluate()
    at ReferencePip
line.collect()
    at FootShootJa
a8.main
</code></pre>
<blockquote>
Since the exception is thrown late, it results in wasted programmer effort se
rching for where the wrong type could have been inserted into the list.
</blockquote>
异常在运行时抛出，而且抛出的位置让程序员很难找到问题所在，浪费了大量的时间。
<blockquote>
And yet there has always been a better way, even in Java 5. We can wrap our List object w
th a checkedList. This way, every time we add an element, it is checked that it is of the correct
ype. The ClassCastException thus happens during the <code>add(42)</code>, rather than m
ch later. For example:
</blockquote>
但是总有办法来解决这个问题，甚至在 java1.5 我们也能进行集合的检查，有种方法可以在数据
入集合时就可以去检查数据类型是否正确，在 <code>add(42)</code> 这一行执行时就可以抛出异
不就是我们想要的？看下面的代码：
<pre><code class="highlight-chroma"><span class="highlight
cl">import java.util.*;
import static java
util.stream.Collectors.*;

```
</span></span><span class="highlight-line"><span class="highlight-cl">public class FootHootWithSafetyCatch {
</span></span><span class="highlight-line"><span class="highlight-cl">    public static void main(String... args) {
</span></span><span class="highlight-line"><span class="highlight-cl">        List&lt;String&gt; names = Collections.checkedList(new ArrayList&lt;String&gt;(), String.class);
</span></span><span class="highlight-line"><span class="highlight-cl">        Collections.addAll(names, "John","Anton","Heinz");
</span></span><span class="highlight-line"><span class="highlight-cl">        List huh = names;
</span></span><span class="highlight-line"><span class="highlight-cl">        List&lt;Integer&gt; numbers = huh;
</span></span><span class="highlight-line"><span class="highlight-cl">        numbers.add(42); //异常将会出现在这一行
</span></span><span class="highlight-line"><span class="highlight-cl">        System.out.println(names.stream().collect(joining("+")));
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<blockquote>
<p>We would still get a ClassCastException, but at the place where the damage was done:</p>
</blockquote>
<p>我们依然会触发一个类型转换异常，但是这次抛出的位置正是错误出现的位置。</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight-cl">ClassCastException: Attempt to insert class Integer element into collection with element type class String
</span></span><span class="highlight-line"><span class="highlight-cl">    at java.util.Collections$CheckedCollection.typeCheck()
</span></span><span class="highlight-line"><span class="highlight-cl">    at java.util.Collections$CheckedCollection.add()
</span></span><span class="highlight-line"><span class="highlight-cl">    at FootShootWithSafetyCatch.main
</span></span></code></pre>
<blockquote>
<p>The checked collection would also discover objects that are added via reflection and throw a ClassCastException. It could not safeguard against "deep reflection", but then not much can.</p>
</blockquote>
<p>这个集合检查功能同样可以用于反射检查，抛出类型转换异常，有兴趣的可以去查阅资料，但不去应付“深度反射”（反射层数较多），但这并不是什么大问题。</p>
<blockquote>
<p>You might wonder why I am writing about a method that was added in Java 5? The reason is that hardly anybody I speak to has heard of <code>Collections.checkedCollection()</code> and its derivatives. It is useful to make your collections just a bit more robust against accidental or deliberate tomfoolery.</p>
</blockquote>
<p>你肯定问为什么我要拿 java 1.5 来讲这个例子，原因是我想强调一下很少有人知道 <code>Collections.checkedCollection()</code> 和他的衍生物，这是在工作中能让我们的代码更加健壮且避免生一些愚蠢的问题和浪费不必要的时间。</p>
<blockquote>
<p>It can also be a quick and easy way to debug any ClassCastException you might discover in your system. Wrap the collection in a checked exception and the guilty party will quickly come to the fore.</p>
```

</blockquote>
这也是一种快速的方式帮助我们找到类型转换异常的方式，把集合包装起来好让问题尽快的暴露来。
<blockquote>
Oh one last thing, completely unrelated to Java, but definitely related to our profession. oday also marks one year since I started my running streak, running at least one mile a day, in snow, rain, lightning and 48C heat. It's been fun and a great way to think about all sorts of thi gs. I've had far more energy, have slept better and have produced more this year than in man previous years. If you're a couch potato, I can only recommend you try Streak Running and j in me in the list of people who've run for at least one year, every day. No excuses.
</blockquote>
oh. 还有一件很重要的事，跟 Java 没关系，但是对我们的职业很有好处，今天已经是我连续跑一年的日子，每天至少跑一公里，不管是刮风下雪，闪电还是 48 度的桑拿天，对我们做事还是工作非常有好处，我觉得我比之前更有能量了，吃的好睡的好，工作状态比前几年要好太多了，如果你是个懒癌患者（couch potato?）我非常的建议你跟我一样加入跑步的行列，坚持一年你将看到非常显的效果。
Kind regards from Crete
Heinz