链滴

使用 Google Guava 快乐编程

作者：jianzh5

原文链接：https://ld246.com/article/1607561493387

来源网站：链滴

目前 Google Guava 在实际应用中非常广泛，本篇博客将以博主对 Guava 使用的认识以及在项中的经验来给大家分享！
正如标题所言，学习使用 Google Guava 可以让你快乐编程，写出优雅的 JAVA 代码！
<h2 id="以面向对象思想处理字符串-Joiner-Splitter-CharMatcher">以面向对象思想处理字符串:Joner/Splitter/CharMatcher</h2>
<blockquote>
JDK 提供的 String 还不够好么？
也许还不够友好，至少让我们用起来还不够爽，还得操心！
举个栗子，比如 String 提供的 split 方法，我们得关心空字符串吧，还得考虑返回的果中存在 null 元素吧，只提供了前后 trim 的方法（如果我想对中间元素进行 trim 呢）。
</blockquote>
那么，看下面的代码示例，guava 让你不必在操心这些：

<blockquote>
Joiner 是连接器，Splitter 是分割器，通常我们会把它们定义为 static final，利用 on 生成对象后在应用到 String 进行处理，这是可以复用的。要知道 apache commons StringUtils 提的都是 static method。更加重要的是，guava 提供的 Joiner/Splitter 是经过充分测试，它的稳定和效率要比 apache 高出不少，这个你可以自行测试下~
发现没有我们想对 String 做什么操作，就是生成自己定制化的 Joiner/Splitter，多么白，简单，流畅的 API！
对于 Joiner，常用的方法是 跳过 NULL 元素：skipNulls() / 对于 NULL 元素使用他替代：useForNull(String)
对于 Splitter，常用的方法是： trimResults()/omitEmptyStrings()。注意拆分的方，有字符串，还有正则，还有固定长度分割（太贴心了！）
</blockquote>
其实除了 Joiner/Splitter 外，guava 还提供了字符串匹配器：CharMatcher
<blockquote>
CharMatcher，将字符的匹配和处理解耦，并提供丰富的方法供你使用！
</blockquote>

<h2 id="对基本类型进行支持">对基本类型进行支持</h2>
<blockquote>
guava 对 JDK 提供的原生类型操作进行了扩展，使得功能更加强大！
</blockquote>

<blockquote>
guava 提供了 Bytes/Shorts/Ints/longs/Floats/Doubles/Chars/Booleans 这些基本据类型的扩展支持，只有你想不到的，没有它没有的！
</blockquote>
<h2 id="对JDK集合的有效补充">对 JDK 集合的有效补充</h2>
<h3 id="灰色地带-Multiset">灰色地带:Multiset</h3>
<blockquote>
JDK 的集合，提供了有序且可以重复的 List，无序且不可以重复的 Set。那这里其实于集合涉及到了 2 个概念，一个 order，一个 dups。那么 List vs Set，and then some ?
</blockquote>

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/12/solofetchupload6362184483659619569-5e590b86.webp?imageView2/2/interlace/1/format/jpg">
<blockquote>
Multiset 是什么，我想上面的图，你应该了解它的概念了。Multiset 就是无序的，但可以重复的集合，它就是游离在 List/Set 之间的“灰色地带”！（至于有序的，不允许重复的集合嘛 guava 还没有提供，当然在未来应该会提供 UniqueList，我猜的，哈哈）
</blockquote>
来看一个 Multiset 的示例：
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/12/solofetchupload5266729228632332505-8675132e.webp?imageView2/2/interlace/1/format/jpg">
<blockquote>
Multiset 自带一个有用的功能，就是可以跟踪每个对象的数量。
</blockquote>
<h3 id="Immutable-vs-unmodifiable">Immutable vs unmodifiable</h3>
来我们先看一个 unmodifiable 的例子：
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/12/solofetchupload1679098560148142756-6320239c.webp?imageView2/2/interlace/1/format/jpg">
<blockquote>
你看到 JDK 提供的 unmodifiable 的缺陷了吗？
实际上，Collections.unmodifiableXxx 所返回的集合和源集合是同一个对象，只不可以对集合做出改变的 API 都被 override，会抛出 UnsupportedOperationException。
也即是说我们改变源集合，导致不可变视图（unmodifiable View）也会发生变化，oh my god!
</blockquote>
当然，在不使用 guava 的情况下，我们是怎么避免上面的问题的呢？
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/12/solofetchupload6926426481937828328-7cf82438.webp?imageView2/2/interlace/1/format/jpg">
<blockquote>
上面揭示了一个概念：Defensive Copies，保护性拷贝。
</blockquote>
OK，unmodifiable 看上去没有问题呢，但是 guava 依然觉得可以改进，于是提出了 Immutabl 的概念，来看：
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/12/solofetchupload2157357132654699459-343a2d28.webp?imageView2/2/interlace/1/format/jpg">
<blockquote>
就一个 copyOf，你不会忘记，如此 cheap~
用 Google 官方的说法是：we're using just one class,just say exactly what we mean，很了不起吗（不仅仅是个概念，Immutable 在 COPY 阶段还考虑了线程的并发性等，很智能的），O(∩_∩)O 哈哈~
guava 提供了很多 Immutable 集合，比如 ImmutableList/ImmutableSet/ImmutableSortedSet/ImmutableMap/.....
</blockquote>
看一个 ImmutableMap 的例子：
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/12/solofetchupload7136330546027268061-d71a2a61.webp?imageView2/2/interlace/1/format/jpg">
<h3 id="可不可以一对多-Multimap">可不可以一对多：Multimap</h3>
<blockquote>

JDK 提供给我们的 Map 是一个键，一个值，一对一的，那么在实际开发中，显然存一个 KEY 多个 VALUE 的情况（比如一个分类下的书本），我们往往这样表达：Map<k,List> 好像有点臃肿！臃肿也就算了，更加不爽的事，我们还得判断 KEY 是否存在来决定是否 new 一个 LIS 出来，有点麻烦！更加麻烦的事情还在后头，比如遍历，比如删除，so hard.....
</blockquote>
来看 guava 如何替你解决这个大麻烦的：

<blockquote>
友情提示下，guava 所有的集合都有 create 方法，这样的好处在于简单，而且我们必在重复泛型信息了。
get()/keys()/keySet()/values()/entries()/asMap()都是非常有用的返回 view collecti n 的方法。
Multimap 的实现类有：ArrayListMultimap/HashMultimap/LinkedHashMultimap TreeMultimap/ImmutableMultimap/.....
</blockquote>
<h3 id="可不可以双向-BiMap">可不可以双向：BiMap</h3>
<blockquote>
JDK 提供的 MAP 让我们可以 find value by key，那么能不能通过 find key by value 呢，能不能 KEY 和 VALUE 都是唯一的呢。这是一个双向的概念，即 forward+backward。</stron >
在实际场景中有这样的需求吗？比如通过用户 ID 找到 mail，也需要通过 mail 找回用名。没有 guava 的时候，我们需要 create forward map AND create backward map，and now ju t let guava do that for you.
</blockquote>

<blockquote>
biMap / biMap.inverse() / biMap.inverse().inverse() 它们是什么关系呢？
你可以稍微看一下 BiMap 的源码实现，实际上，当你创建 BiMap 的时候，在内部维了 2 个 map，一个 forward map，一个 backward map，并且设置了它们之间的关系。
因此，biMap.inverse() != biMap ； biMap.inverse().inverse() == biMap</stron >
</blockquote>
<h3 id="可不可以多个KEY-Table">可不可以多个 KEY：Table</h3>
<blockquote>
我们知道数据库除了主键外，还提供了复合索引，而且实际中这样的多级关系查找也比较多的，当然我们可以利用嵌套的 Map 来实现：Map<k1,Map<k2,v2>>。为了让我的代码看起来不那么丑陋，guava 为我们提供了 Table。
</blockquote>

<blockquote>
Table 涉及到 3 个概念：rowKey,columnKey,value，并提供了多种视图以及操作方让你更加轻松的处理多个 KEY 的场景。
</blockquote>
<hr>
<h2 id="函数式编程-Functions">函数式编程：Functions</h2>

com/file/2020/12/solofetchupload3444500289433635524-8e7e4845.webp?imageView2/2/interlace/1/format/jpg">
<blockquote>
上面的代码是为了完成将 List 集合中的元素，先截取 5 个长度，然后转成大写。
函数式编程的好处在于在集合遍历操作中提供自定义 Function 的操作，比如 transform 转换。我们再也不需要一遍遍的遍历集合，显著的简化了代码！
</blockquote>
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/12/solofetchupload7644422163408785542-8b35abb5.webp?imageView2/2/interlace/1/format/jpg">
<h2 id="断言-Predicate">断言：Predicate<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/12/solofetchupload166402719894803971-ffc9d8ef.webp?imageView2/2/interlace/1/format/jpg"></h2>
<blockquote>
Predicate 最常用的功能就是运用在集合的过滤当中！
</blockquote>
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/12/solofetchupload5588225641173567787-4a09fe32.webp?imageView2/2/interlace/1/format/jpg">
<blockquote>
需要注意的是 Lists 并没有提供 filter 方法，不过你可以使用 Collections2.filter 完成
</blockquote>
<hr>
<h2 id="check-null-and-other-Optional-Preconditions">check null and other：Optional、Preonditions</h2>
在 guava 中，对于 null 的处理手段是快速失败，你可以看看 guava 的源码，很多方的第一行就是：Preconditions.checkNotNull(elements);
要知道 null 是模糊的概念，是成功呢，还是失败呢，还是别的什么含义呢？
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/12/solofetchupload7419731527347178279-6d8384bc.webp?imageView2/2/interlace/1/format/jpg">
<h2 id="Cache-is-king">Cache is king</h2>
<blockquote>
对于大多数互联网项目而言，缓存的重要性，不言而喻！
如果我们的应用系统，并不想使用一些第三方缓存组件（如 redis），我们仅仅想在本有一个功能足够强大的缓存，很可惜 JDK 提供的那些 SET/MAP 还不行！
</blockquote>
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/12/solofetchupload4283178100605392919-d81d5f9f.webp?imageView2/2/interlace/1/format/jpg">
<blockquote>
首先，这是一个本地缓存，guava 提供的 cache 是一个简洁、高效，易于维护的。为么这么说呢？因为并没有一个单独的线程用于刷新 OR 清理 cache，对于 cache 的操作，都是通过问/读写带来的，也就是说在读写中完成缓存的刷新操作！
其次，我们看到了，我们非常通俗的告诉 cache，我们的缓存策略是什么，SO EASY 在如此简单的背后，是 guava 帮助我们做了很多事情，比如线程安全。
</blockquote>
<h2 id="让异步回调更加简单">让异步回调更加简单</h2>
<blockquote>
JDK 中提供了 Future/FutureTask/Callable 来对异步回调进行支持，但是还是看上去复杂的，能不能更加简单呢？比如注册一个监听回调。
</blockquote>

</blockquote>
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/12/solofetchupload1395797098073448635-82f79434.webp?imageView2/2/interlace/1/format/jpg">
<blockquote>
我们可以通过 guava 对 JDK 提供的线程池进行装饰，让其具有异步回调监听功能，后在设置监听器即可！
</blockquote>
<h2 id="Summary">Summary</h2>
到这里，这篇文章也只介绍了 guava 的冰山一角，其实还有很多内容，比如反射、注解、网络并发、IO 等等
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/12/solofetchupload5963137669773690501-49e12919.webp?imageView2/2/interlace/1/format/jpg">