**Google Guava Collections 使用介绍**

Google Guava Collections（以下都简称为 Guava Collections）是 Java Collections Framework 的增强和扩展。每个 Java 开发者都会在工作中使用各种数据结构，很多情况下 Java Collections Framework 可以帮助你完成这类工作。但是在有些场合你使用了 Java Collections Framework 的 API，但还是需要写很多代码来实现一些复杂逻辑，这个时候就可以尝试使用 Guava Collections 来帮助你完成这些工作。这些高质量的 API 使你的代码更短，更易于阅读和修改，工作更加轻松。

**目标读者**

对于理解 Java 开源工具来说，本文读者至少应具备基础的 Java 知识，特别是 JDK5 的特性。因为 Guava Collections 充分使用了范型，循环增强这样的特性。作为 Java Collections Framework 的增强，读者必须对 Java Collections Framework 有清晰的理解，包括主要的接口约定和常用的实现类。并且 Guava Collections 很大程度上是帮助开发者完成比较复杂的数据结构的操作，因此基础的数据结构和算法的知识也是清晰理解 Guava Collections 的必要条件。

**项目背景**

Guava Collections 是 Google 的工程师 Kevin Bourrillion 和 Jared Levy 在著名"20%"时间写的代码。当然作为开源项目还有其他的开发者贡献了代码。在编写的过程中，Java Collections Framework 的作者 Joshua Bloch 也参与了代码审核和提出建议。目前它已经移到另外一个叫 guava-libraries 的开源项目下面来维护。

因为功能相似而且又同是开源项目，人们很很自然会把它和 Apache Commons Collections 来做比较。其区别归结起来有以下几点：

Guava Collections 充分利用了 JDK5 的范型和枚举这样的特性，而 Apache Commons Collections 则是基于 JDK1.2。其次 Guava Collections 更加严格遵守 Java Collections Framework 定义的接口契约，而在 Apache Commons Collections 你会发现不少违反这些 JDK 接口契约的地方。这些不遵守标准的地方就是出 bug 的风险很高。最后 Guava Collections 处于积极的维护状态，本文介绍的特性都基于目前最新 2011 年 4 月的 Guava r09 版本，而 Apache Commons Collections 最新一次发布也已经是 2008 年了。

**功能列举**

可以说 Java Collections Framework 满足了我们大多数情况下使用集合的要求，但是当遇到一些特殊的情况我们的代码会比较冗长，比较容易出错。Guava Collections 可以帮助你的代码更简短精炼，更重要是它增强了代码的可读性。看看 Guava Collections 为我们做了哪些很酷的事情。

* Immutable Collections: 还在使用 Collections.unmodifiableXXX() ？ Immutable Collections 这才是真正的不可修改的集合
* Multiset: 看看如何把重复的元素放入一个集合
* Multimaps: 需要在一个 key 对应多个 value 的时候 , 自己写一个实现比较繁琐 - 让 Multimaps 来帮忙
* BiMap: java.util.Map 只能保证 key 的不重复，BiMap 保证 value 也不重复
* MapMaker: 超级强大的 Map 构造类
* Ordering class: 大家知道用 Comparator 作为比较器来对集合排序，但是对于多关键字排序 Ordering class 可以简化很多的代码
* 其他特性

当然，如果没有 Guava Collections 你也可以用 Java Collections Framework 完成上面的功能。但是 Guava Collections 提供的这些 API 经过精心设计，而且还有 25000 个单元测试来保障它的质量。所以我们没必要重新发明轮子。接下来我们来详细看看 Guava Collections 的一些具体功能。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-googlecollection/#ibm-pcon)

**Immutable Collections: 真正的不可修改的集合**

大家都用过 Collections.unmodifiableXXX() 来做一个不可修改的集合。例如你要构造存储常量的 Set，你可以这样来做 :

|  |
| --- |
| Set<String> set = new HashSet<String>(Arrays.asList(new String[]{"RED", "GREEN"}));  Set<String> unmodifiableSet = Collections.unmodifiableSet(set); |

这看上去似乎不错，因为每次调 unmodifiableSet.add() 都会抛出一个 UnsupportedOperationException。感觉安全了？慢！如果有人在原来的 set 上 add 或者 remove 元素会怎么样？结果 unmodifiableSet 也是被 add 或者 remove 元素了。而且构造这样一个简单的 set 写了两句长的代码。下面看看 ImmutableSet 是怎么来做地更安全和简洁 :

|  |
| --- |
| ImmutableSet<String> immutableSet = ImmutableSet.of("RED", "GREEN"); |

就这样一句就够了，而且试图调 add 方法的时候，它一样会抛出 UnsupportedOperationException。重要的是代码的可读性增强了不少，非常直观地展现了代码的用意。如果像之前这个代码保护一个 set 怎么做呢？你可以 :

|  |
| --- |
| ImmutableSet<String> immutableSet = ImmutableSet.copyOf(set); |

从构造的方式来说，ImmutableSet 集合还提供了 Builder 模式来构造一个集合 :

|  |
| --- |
| Builder<String> builder = ImmutableSet.builder();  ImmutableSet<String> immutableSet = builder.add("RED").addAll(set).build(); |

在这个例子里面 Builder 不但能加入单个元素还能加入既有的集合。

除此之外，Guava Collections 还提供了各种 Immutable 集合的实现：ImmutableList，ImmutableMap，ImmutableSortedSet，ImmutableSortedMap。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-googlecollection/#ibm-pcon)

**Multiset: 把重复的元素放入集合**

你可能会说这和 Set 接口的契约冲突，因为 Set 接口的 JavaDoc 里面规定不能放入重复元素。事实上，Multiset 并没有实现 java.util.Set 接口，它更像是一个 Bag。普通的 Set 就像这样 :[car, ship, bike]，而 Multiset 会是这样 : [car x 2, ship x 6, bike x 3]。

譬如一个 List 里面有各种字符串，然后你要统计每个字符串在 List 里面出现的次数 :

|  |
| --- |
| Map<String, Integer> map = new HashMap<String, Integer>();  for(String word : wordList){  Integer count = map.get(word);  map.put(word, (count == null) ? 1 : count + 1);  }  //count word “the”  Integer count = map.get(“the”); |

如果用 Multiset 就可以这样 :

|  |
| --- |
| HashMultiset<String> multiSet = HashMultiset.create();  multiSet.addAll(wordList);  //count word “the”  Integer count = multiSet.count(“the”); |

这样连循环都不用了，而且 Multiset 用的方法叫 count，显然比在 Map 里面调 get 有更好的可读性。Multiset 还提供了 setCount 这样设定元素重复次数的方法，虽然你可以通过使用 Map 来实现类似的功能，但是程序的可读性比 Multiset 差了很多。

常用实现 Multiset 接口的类有：

* HashMultiset: 元素存放于 HashMap
* LinkedHashMultiset: 元素存放于 LinkedHashMap，即元素的排列顺序由第一次放入的顺序决定
* TreeMultiset:元素被排序存放于TreeMap
* EnumMultiset: 元素必须是 enum 类型
* ImmutableMultiset: 不可修改的 Mutiset

看到这里你可能已经发现 Guava Collections 都是以 create 或是 of 这样的静态方法来构造对象。这是因为这些集合类大多有多个参数的私有构造方法，由于参数数目很多，客户代码程序员使用起来就很不方便。而且以这种方式可以返回原类型的子类型对象。另外，对于创建范型对象来讲，这种方式更加简洁。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-googlecollection/#ibm-pcon)

**Multimap: 在 Map 的 value 里面放多个元素**

Muitimap 就是一个 key 对应多个 value 的数据结构。看上去它很像 java.util.Map 的结构，但是 Muitimap 不是 Map，没有实现 Map 的接口。设想你对 Map 调了 2 次参数 key 一样的 put 方法，结果就是第 2 次的 value 覆盖了第 1 次的 value。但是对 Muitimap 来说这个 key 同时对应了 2 个 value。所以 Map 看上去是 : {k1=v1, k2=v2,...}，而 Muitimap 是 :{k1=[v1, v2, v3], k2=[v7, v8],....}。

举个记名投票的例子。所有选票都放在一个 List<Ticket> 里面，List 的每个元素包括投票人和选举人的名字。我们可以这样写 :

|  |
| --- |
| //Key is candidate name, its value is his voters  HashMap<String, HashSet<String>> hMap = new HashMap<String, HashSet<String>>();  for(Ticket ticket: tickets){  HashSet<String> set = hMap.get(ticket.getCandidate());  if(set == null){  set = new HashSet<String>();  hMap.put(ticket.getCandidate(), set);  }  set.add(ticket.getVoter());  } |

我们再来看看 Muitimap 能做些什么 :

|  |
| --- |
| HashMultimap<String, String> map = HashMultimap.create();  for(Ticket ticket: tickets){  map.put(ticket.getCandidate(), ticket.getVoter());  } |

就这么简单！

Muitimap 接口的主要实现类有：

* HashMultimap: key 放在 HashMap，而 value 放在 HashSet，即一个 key 对应的 value 不可重复
* ArrayListMultimap: key 放在 HashMap，而 value 放在 ArrayList，即一个 key 对应的 value 有顺序可重复
* LinkedHashMultimap: key 放在 LinkedHashMap，而 value 放在 LinkedHashSet，即一个 key 对应的 value 有顺序不可重复
* TreeMultimap: key 放在 TreeMap，而 value 放在 TreeSet，即一个 key 对应的 value 有排列顺序
* ImmutableMultimap: 不可修改的 Multimap

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-googlecollection/#ibm-pcon)

**BiMap: 双向 Map**

BiMap 实现了 java.util.Map 接口。它的特点是它的 value 和它 key 一样也是不可重复的，换句话说它的 key 和 value 是等价的。如果你往 BiMap 的 value 里面放了重复的元素，就会得到 IllegalArgumentException。

举个例子，你可能经常会碰到在 Map 里面根据 value 值来反推它的 key 值的逻辑：

|  |
| --- |
| for(Map.Entry<User, Address> entry : map.entreSet()){  if(entry.getValue().equals(anAddess)){  return entry.getKey();  }  }  return null; |

如果把 User 和 Address 都放在 BiMap，那么一句代码就得到结果了：

|  |
| --- |
| return biMap.inverse().get(anAddess); |

这里的 inverse 方法就是把 BiMap 的 key 集合 value 集合对调，因此 biMap == biMap.inverse().inverse()。

BiMap的常用实现有：

HashBiMap: key 集合与 value 集合都有 HashMap 实现

EnumBiMap: key 与 value 都必须是 enum 类型

ImmutableBiMap: 不可修改的 BiMap

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-googlecollection/#ibm-pcon)

**MapMaker: 超级强大的 Map 构造工具**

MapMaker 是用来构造 ConcurrentMap 的工具类。为什么可以把 MapMaker 叫做超级强大？看了下面的例子你就知道了。首先，它可以用来构造 ConcurrentHashMap:

|  |
| --- |
| //ConcurrentHashMap with concurrency level 8  ConcurrentMap<String, Object> map1 = new MapMaker()  .concurrencyLevel(8)  .makeMap(); |

或者构造用各种不同 reference 作为 key 和 value 的 Map:

|  |
| --- |
| //ConcurrentMap with soft reference key and weak reference value  ConcurrentMap<String, Object> map2 = new MapMaker()  .softKeys()  .weakValues()  .makeMap(); |

或者构造有自动移除时间过期项的 Map:

|  |
| --- |
| //Automatically removed entries from map after 30 seconds since they are created  ConcurrentMap<String, Object> map3 = new MapMaker()  .expireAfterWrite(30, TimeUnit.SECONDS)  .makeMap(); |

或者构造有最大限制数目的 Map：

|  |
| --- |
| //Map size grows close to the 100, the map will evict  //entries that are less likely to be used again  ConcurrentMap<String, Object> map4 = new MapMaker()  .maximumSize(100)  .makeMap(); |

或者提供当 Map 里面不包含所 get 的项，而需要自动加入到 Map 的功能。这个功能当 Map 作为缓存的时候很有用 :

|  |
| --- |
| //Create an Object to the map, when get() is missing in map  ConcurrentMap<String, Object> map5 = new MapMaker()  .makeComputingMap(  new Function<String, Object>() {  public Object apply(String key) {  return createObject(key);  }}); |

这些还不是最强大的特性，最厉害的是 MapMaker 可以提供拥有以上所有特性的 Map:

|  |
| --- |
| //Put all features together!  ConcurrentMap<String, Object> mapAll = new MapMaker()  .concurrencyLevel(8)  .softKeys()  .weakValues()  .expireAfterWrite(30, TimeUnit.SECONDS)  .maximumSize(100)  .makeComputingMap(  new Function<String, Object>() {  public Object apply(String key) {  return createObject(key);  }}); |

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-googlecollection/#ibm-pcon)

**Ordering class: 灵活的多字段排序比较器**

要对集合排序或者求最大值最小值，首推 java.util.Collections 类，但关键是要提供 Comparator 接口的实现。假设有个待排序的 List<Foo>，而 Foo 里面有两个排序关键字 int a, int b 和 int c:

|  |
| --- |
| Collections.sort(list, new Comparator<Foo>(){  @Override  public int compare(Foo f1, Foo f2) {  int resultA = f1.a – f2.a;  int resultB = f1.b – f2.b;  return resultA == 0 ? (resultB == 0 ? f1.c – f2.c : resultB) : resultA; |

}});

这看上去有点眼晕，如果用一串 if-else 也好不到哪里去。看看 ComparisonChain 能做到什么 :

|  |
| --- |
| Collections.sort(list, new Comparator<Foo>(){  @Override  return ComparisonChain.start()  .compare(f1.a, f2.a)  .compare(f1.b, f2.b)  .compare(f1.c, f2.c).result();  }}); |

如果排序关键字要用自定义比较器，compare 方法也有接受 Comparator 的重载版本。譬如 Foo 里面每个排序关键字都已经有了各自的 Comparator，那么利用 ComparisonChain 可以 :

|  |
| --- |
| Collections.sort(list, new Comparator<Foo>(){  @Override  return ComparisonChain.start()  .compare(f1.a, f2.a, comparatorA)  .compare(f1.b, f2.b, comparatorB)  .compare(f1.c, f2.c, comparatorC).result();  }}); |

Ordring 类还提供了一个组合 Comparator 对象的方法。而且 Ordring 本身实现了 Comparator 接口所以它能直接作为 Comparator 使用：

|  |
| --- |
| Ordering<Foo> ordering = Ordering.compound(\  Arrays.asList(comparatorA, comparatorB, comparatorc));  Collections.sort(list, ordering); |

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-googlecollection/#ibm-pcon)

**其他特性 :**

过滤器：利用 Collections2.filter() 方法过滤集合中不符合条件的元素。譬如过滤一个 List<Integer> 里面小于 10 的元素 :

|  |
| --- |
| Collection<Integer> filterCollection =  Collections2.filter(list, new Predicate<Integer>(){  @Override  public boolean apply(Integer input) {  return input >= 10;  }}); |

当然，你可以自己写一个循环来实现这个功能，但是这样不能保证之后小于 10 的元素不被放入集合。filter 的强大之处在于返回的 filterCollection 仍然有排斥小于 10 的元素的特性，如果调 filterCollection.add(9) 就会得到一个 IllegalArgumentException。

转换器：利用 Collections2.transform() 方法来转换集合中的元素。譬如把一个 Set<Integer> 里面所有元素都转换成带格式的 String 来产生新的 Collection<String>:

|  |
| --- |
| Collection<String> formatCollection =  Collections2.transform(set, new Function<Integer, String>(){  @Override  public String apply(Integer input) {  return new DecimalFormat("#,###").format(input);  }} ); |

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-googlecollection/#ibm-pcon)

**下载与使用**

这个开源项目发布的 jar 包可以在它的官方网站内（<http://code.google.com/p/guava-libraries/downloads/list>）找到。其下载的 zip 包中含有 Guava Collections 的 jar 包 guava-r09.jar 及其依赖包 guava-r09-gwt.jar，javadoc，源代码，readme 等文件。使用时只需将 guava-r09.jar 和依赖包 guava-r09-gwt.jar 放入 CLASSPATH 中即可。

如果您使用 Maven 作为构建工具的话可以在 pom.xml 内加入：

|  |
| --- |
| <dependency>  <groupId>com.google.guava</groupId>  <artifactId>guava</artifactId>  <version>r09</version>  </dependency> |

需要注意的是本文介绍的 Guava r09 需要 1.5 或者更高版本的 JDK。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-googlecollection/#ibm-pcon)

**结束语**

以上介绍了 Guava Collections 的一些基本的功能特性。你可以从 guava-libraries 的官方网站下载它的 jar 包和它其他的相关文档。如果你使用 Maven 来管理你的项目依赖包，Maven 中央库也提供了它版本的依赖。最后希望 Guava Collections 使你的编程工作更轻松，更有乐趣。