谷歌 guava 项目分析

张宁鑫
2016K8009915005
github: elxe

【项目简介】

谷歌 guava 可以大致分为三个部分：用基础的实用程序去减少实现常见方法和行为的琐事(basic utilities to reduce menial labors to implement common methods and behaviors)；以前称为 Google Collections Library 的 Java 集合框架 (JCF) 的扩展；以及其他提供方便和高效功能的实用程序，如函数式编程，图形，缓存，范围对象和散列。集合组件的创建和体系结构部分是由 JDK1.5 中引入的泛型驱动的。尽管泛型提高了程序员的工作效率，但标准 JCF 并没有提供足够的功能，而且它的补充 Apache Commons Collections 没有采用泛型来保持向后兼容性。这一事实促使两位工程师 Kevin Bourrillion 和 Jared Levy 开发了 JCF 扩展，后者提供了额外的泛型类，如多集，多图，bimaps 以及不可变集合。

Java Collections 框架的原始首席设计师 Joshua Bloch 和 JDK 中并发实用程序的首席设计师之一 Doug Lea 建议并审阅了库的设计和代码。

截至 2012 年 4 月，Guava 排名第 12 位最受欢迎的 Java 库，仅次于 Apache Commons 项目和其他一些项目。2013 年对 10,000 个 GitHub 项目进行的研究发现，谷歌制造的库，如 Google Web Toolkit 和 Guava，构成了 Java 中最受欢迎的 100 个最受欢迎的图书馆中的 7 个，而 Guava 是第 8 个最受欢迎的 Java 图书馆。截至 2018 年 3 月，Guava 是 Github 上排名第六的 Java 项目。

【不可变集合功能分析】

◆ **不可变集合（Immutable collections）**

特点：

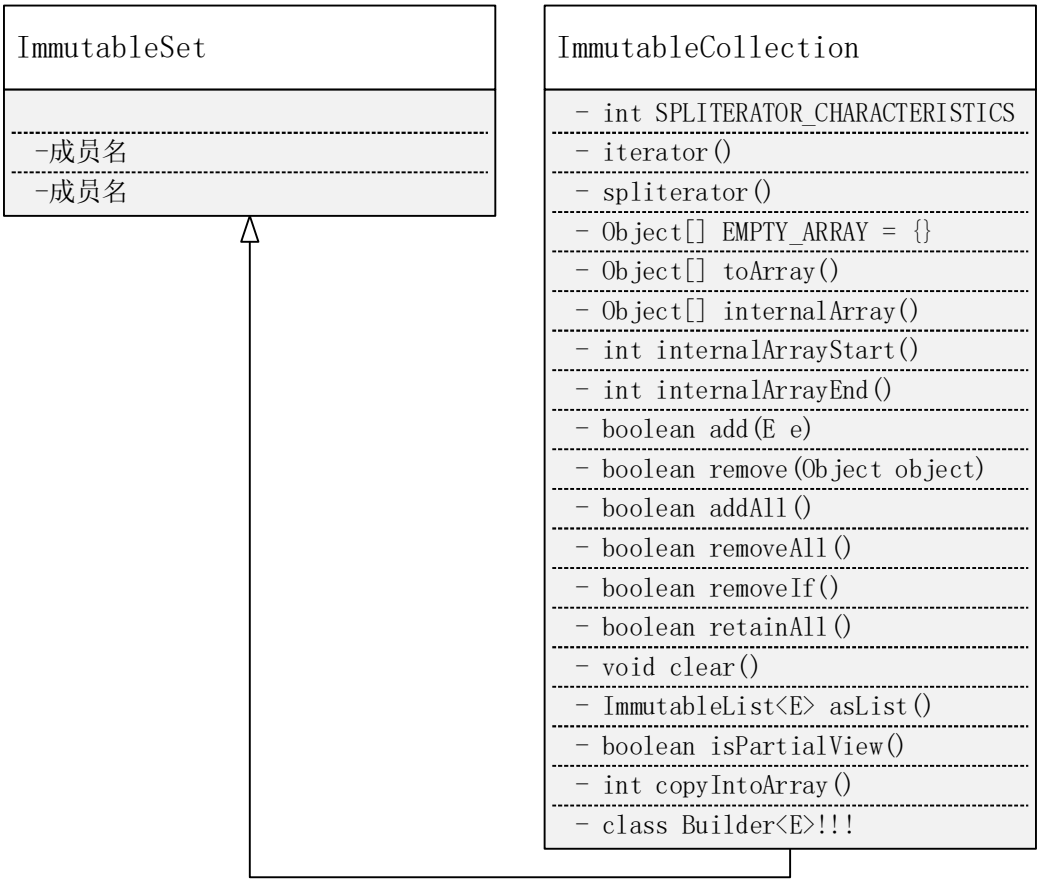
- 不受信任的库可以安全使用。
- 线程安全：可以被许多线程使用，没有竞争条件的风险。
- 不需要支持变异，并且可以通过该假设节省时间和空间。所有不可变的集合实现比它们的可变兄弟节点更具内存效率。(分析)
- 可以用作常量，期望它将保持固定。

示例：

```java
public static final ImmutableSet<String> COLOR_NAMES = ImmutableSet.of(
        "red",
        "orange",
        "yellow",
        "green",
        "blue",
        "purple");

class Foo {
        final ImmutableSet<Bar> bars;
        Foo(Set<Bar> bars) {
            this.bars = ImmutableSet.copyOf(bars); // defensive copy!
        }
}
```

建模:



(未完待续)

- ◆ **新集合类型（New Collection Types）**
  - **多集**

    定义:

    维基百科将数学中的多重集合定义为“集合概念的概括，其中成员被允许不止

一次出现……在多集合中，如集合中，与元组相反，元素的顺序无关紧要：multisets {a，a，b}和{a，b，a}相等。

优势：

Guava 的 MultisetAPI 结合了两种思考方式 Multiset，如下所示：

- 当被视为正常时 Collection，Multiset 表现得很像无序 ArrayList：
  - 调用 add(E)会添加给定元素的单个匹配项。
  - 在 iterator()一个多重遍历每个元素的每一次出现的。
  - 在 size()一个多重的是所有元件的所有出现的总次数。
- 额外的查询操作以及性能特征就像您期望的那样 Map<E, Integer>。
  - count(Object)返回与该元素关联的计数。对于 a HashMultiset，count 为 O（1），对于 a TreeMultiset，count 为 O（log n）等。
  - entrySet()返回一个 Set<Multiset.Entry<E>>类似于 a 的 entrySet 的东西 Map。
  - elementSet()返回 Set<E>多重集的不同元素之一，就像 keySet()a 一样 Map。
  - 实现的内存消耗 Multiset 在不同元素的数量上是线性的。

示例：

之前常用的统计文档单词数的程序：

```
Map<String, Integer> counts = new HashMap<String, Integer>();
       for (String word : words) {
         Integer count = counts.get(word);
         if (count == null) {
       counts.put(word, 1);
         } else {
           counts.put(word, count + 1);
       }}
```

如果用多集可以调用 count(Object)返回与该元素关联的计数，更好的解决这个问题。

- **SortedMultiset**

定义：

SortedMultiset 是 Multiset 接口的变体，支持在指定范围内有效地获取子多重集。

- **Multimap：**

定义：

Guava 的 Multimap 框架可以轻松处理从键到多个值的映射。A Multimap 是

将密钥与任意多个值相关联的一般方法。

使用情况：

从概念上讲，有两种方法可以考虑 Multimap：作为从单个键到单个值的映射集合：

a -> 1
a -> 2
a -> 4
b -> 3
c -> 5

或者作为从唯一键到值集合的映射：

a -> [1, 2, 4]
b -> [3]
c -> [5]

基本操作：

```
//使用树键和数组列表值

ListMultimap<String, Integer> treeListMultimap =

MultimapBuilder.treeKeys().arrayListValues().build();

//使用哈希键和枚举设置值
SetMultimap<Integer, MyEnum> hashEnumMultimap =

MultimapBuilder.hashKeys().enumSetValues(MyEnum.class).build();
Set<Person> aliceChildren = childrenMultimap.get(alice);
aliceChildren.clear();
aliceChildren.add(bob);
aliceChildren.add(carol);
//添加从键到值的关联。
multimap.get(key).add(value);
//依次将键中的关联添加到每个值
Iterables.addAll(multimap.get(key), values);
//从中删除一个关联 key，value 并 true 在多图更改时返回。
multimap.get(key).remove(value);
//删除并返回与指定键关联的所有值。返回的集合可能是也可能不是可修改的，但修改它不会影响多图。（返回适当的集合类型。）
multimap.get(key).clear();
//清除与之关联的所有值，key 并设置 key 与每个值相关联 values。返回先前与键关联的值。
multimap.get(key).clear();
Iterables.addAll(multimap.get(key), values);
```

- **BIMAP：**

  优势：

  A BiMap<K, V> is a Map<K, V> that

  - 使你可以看它的逆映射 BiMap<V, K> 通过函数 inverse()
  - 确保值都是唯一的, making values() a Set

  示例：

```
Map<String, Integer> nameToId = Maps.newHashMap();
Map<Integer, String> idToName = Maps.newHashMap();
nameToId.put("Bob", 42);
idToName.put(42, "Bob");
//如果“Bob”或 42 已经存在会怎么样?
//如果我们忘记让它们保持同步，就会出现奇怪的错误.....
BiMap<String, Integer> userId = HashBiMap.create();
        ...
String userForId = userId.inverse().get(id);
```

- **Table：**

优势：

Table 它支持任何“行”类型和“列”类型的用例。Table 支持许多视图，让您可以从任何角度使用数据，包括

- rowMap()，把 Table<R, C, V>作为 Map<R, Map<C, V>>展示。同样，rowKeySet() 返回一个 Set<R>。
- row(r)返回一个非 null Map<C, V>。写入 Map 将写入底层 Table。
- 类似的列方法：columnMap()，columnKeySet()，和 column(c)。（基于列的访问比基于行的访问效率稍差）
- cellSet()返回的一个视图 Table 作为一组 Table.Cell<R, C, V>。Cell 很像 Map.Entry，但区分行和列键。

实现方式：

- HashBasedTable，基本上由一个支持 HashMap<R, HashMap<C, V>>。
- TreeBasedTable，基本上由一个支持 TreeMap<R, TreeMap<C, V>>。
- ImmutableTable
- ArrayTable，这要求在构造时指定完整的行和列的范围，但是由二维数组支持，以在表密集时提高速度和内存效率。
- ArrayTable 与其他实现有些不同；有关详细信息，请咨询 Javadoc。

- **ClassToInstanceMap**

优势：

ClassToInstanceMap 提供了一种是用 Class 作为 Key，对应实例作为 Value 的途径.他定义了 T getInstance(Class<T>)和 T putInstance(Class<T> T)两个方法，这两个方法消除了元素类型转换的过程并保证了元素在 Map 中是类型安全的.

特点：

ClassToInstanceMap 有一个独立的类型参数，一般命名为 B. 它对应着 Map 的元素的类型的最大上界.例如

```
ClassToInstanceMap<Number> numberDefaults = MutableClassToInstanceMap.create();
numberDefaults.putInstance(Integer.class, Integer.valueOf(0));
```

实现上, ClassToInstanceMap<B>实现了 Map<Class<? extends B>, B>；换句话说，他是一个由 B 的子类和 B 的实例构成的 Map.

Guava 提供了很有用的 ClassToInstanceMap 的实现 MutableClassToInstanceMap 和 ImmutableClassToInstanceMap

重点:

就像其他 Map<Class, Object>, ClassToInstanceMap 可能会包含原生类型的元素，原生类型和它的包装类在 map 中可能会映射到不同的值上.但是在 getInstance 取值的时候会将所有原生类型都转成它的包装类.

- **RangeSet：**

定义：

A RangeSet 描述了一组断开的非空范围。将范围添加到 mutable 时 RangeSet，任何连接的范围将合并在一起，并忽略空范围。

基本操作：

- complement()：观看补充 RangeSet。complement 也是一个 RangeSet，因为它包含断开的非空范围。
- subRangeSet(Range<C>)：返回 RangeSet 与指定的交集的视图 Range。这概括了 headSet，subSet 和 tailSet 景色的传统排序的集合。
- asRanges()：将视图 RangeSet 视为 Set<Range<C>>可以迭代的视图。
- asSet(DiscreteDomain<C>)（ImmutableRangeSet 仅）：将视图 RangeSet<C>视为一个 ImmutableSortedSet<C>，查看范围中的元素而不是范围本身。（该操作是不支持的，如果 DiscreteDomain 和 RangeSet 是上述二者无界或以下两者无界的。）
- contains(C)：a 上最基本的操作 RangeSet，查询是否 RangeSet 包含指定元素中的任何范围。
- rangeContaining(C)：返回 Range 包含指定元素的内容，或者 null 如果没有则返回。
- encloses(Range<C>)：直截了当不够，测试如果任何 Range 在 RangeSet 封闭了规定的范围。
- span()：返回最小 Range 的是 encloses 每一个范围在此 RangeSet。

◆ **未完待续**