(http://baeldung.com)

¿Cómo almacenar claves duplicadas en un mapa en Java?

Última modificación: 30 de abril de 2018

por Andrea Ligios (http://www.baeldung.com/author/andrea-ligios/)
(http://www.baeldung.com/author/andrea-ligios/)

Guayaba (http://www.baeldung.com/category/guava/)

Java (http://www.baeldung.com/category/java/) +

Colecciones de Java (http://www.baeldung.com/tag/collections/)

1. Información general

En este tutorial, vamos a explorar las opciones disponibles para manejar un *mapa* con claves duplicadas o, en otras palabras, un *mapa* que permite almacenar múltiples valores para una sola tecla.

2. Mapas estándar

Java tiene varias implementaciones de la interfaz *Mapa* , cada una con sus particularidades.

Sin embargo, **ninguna de las implementaciones de Java core Map existentes permite que *Map* maneje múltiples valores para una sola clave** .

Como podemos ver, si tratamos de insertar dos valores para la misma clave, el segundo valor se almacenará, mientras que el primero se descartará.

También será devuelto (por cada implementación adecuada del método *put (clave K, valor V)* *(https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#put-K-V-)*):

```
1 Map<String, String> map = new HashMap<>();
2 assertThat(map.put("key1", "value1")).isEqualTo(null);
3 assertThat(map.put("key1", "value2")).isEqualTo("value1");
4 assertThat(map.get("key1")).isEqualTo("value2");
```

¿Cómo podemos lograr el comportamiento deseado, entonces?

3. Colección como valor

Obviamente, usar una *Colección* para cada valor de nuestro *Mapa* haría el trabajo:

```
1 Map<String, List<String>> map = new HashMap<>();
2 List<String> list = new ArrayList<>();
3 map.put("key1", list);
4 map.get("key1").add("value1");
5 map.get("key1").add("value2");
6
7 assertThat(map.get("key1").get(0)).isEqualTo("value1");
8 assertThat(map.get("key1").get(1)).isEqualTo("value2");
```


Sin embargo, esta solución detallada tiene múltiples inconvenientes y es propensa a errores. Implica que necesitamos crear una instancia de una *Colección* para cada valor, verificar su presencia antes de agregar o eliminar un valor, eliminarla manualmente cuando no quedan valores, etcétera.

Desde Java 8 podríamos explotar los métodos *compute ()* y mejorarlo:

```
Map<String, List<String>> map = new HashMap<>();
map.computeIfAbsent("key1", k -> new ArrayList<>()).add("value1");
map.computeIfAbsent("key1", k -> new ArrayList<>()).add("value2");

assertThat(map.get("key1").get(0)).isEqualTo("value1");
assertThat(map.get("key1").get(1)).isEqualTo("value2");
```

Si bien esto es algo que vale la pena saber, debemos evitarlo a menos que tenga una muy buena razón para no hacerlo, al igual que las políticas restrictivas de la compañía que nos impiden utilizar las bibliotecas de terceros.

De lo contrario, antes de escribir nuestra propia implementación personalizada de *mapas* y reinventar la rueda, debemos elegir entre las varias opciones disponibles listas para usar.

4. Colecciones de Apache Commons

Como de costumbre, *Apache* tiene una solución para nuestro problema.

Comencemos por importar la última versión de *Common Collections* (CC a partir de ahora):

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-collections4</artifactId>
  <version>4.1</version>
</dependency>
```

4.1. MultiMap

El *org.apache.commons.collections4.* ***La*** *(https://commons.apache.org/proper/commons-collections/apidocs/org/apache/commons/collections4/MultiMap.html)* interfaz ***MultiMap*** *(https://commons.apache.org/proper/commons-collections/apidocs/org/apache/commons/collections4/MultiMap.html)* define un Mapa que contiene una colección de valores en cada clave.

Es implementado por *org.apache.commons.collections4.map.* *(https://commons.apache.org/proper/commons-collections/apidocs/org/apache/commons/collections4/map/MultiValueMap.html)*Clase ***MultiValueMap*** *(https://commons.apache.org/proper/commons-*

collections/apidocs/org/apache/commons/collections4/map/MultiValueMap.html) , que maneja automáticamente la mayor parte del texto estándar bajo el capó:

```
MultiMap<String, String> map = new MultiValueMap<>();
map.put("key1", "value1");
map.put("key1", "value2");
assertThat((Collection<String>) map.get("key1"))
  .contains("value1", "value2");
```

Si bien esta clase está disponible desde CC 3.2, **no es segura para subprocesos** y **ha quedado obsoleta en CC 4.1** . Deberíamos usarlo solo cuando no podamos actualizar a la versión más nueva.

4.2. *MultiValuedMapa*

El sucesor de *MultiMap* es *org.apache.commons.collections4. (https://commons.apache.org/proper/commons-collections/apidocs/org/apache/commons/collections4/MultiValuedMap.html)*Interfaz ***MultiValuedMap*** *(https://commons.apache.org/proper/commons-collections/apidocs/org/apache/commons/collections4/MultiValuedMap.html)* . Tiene múltiples implementaciones listas para ser usadas.

Veamos cómo almacenar nuestros múltiples valores en *ArrayList* , que conserva duplicados:

```
MultiValuedMap<String, String> map = new ArrayListValuedHashMap<>();
map.put("key1", "value1");
map.put("key1", "value2");
map.put("key1", "value2");
assertThat((Collection<String>) map.get("key1"))
  .containsExactly("value1", "value2", "value2");
```


Alternativamente, podríamos usar un *HashSet* , que arroja duplicados:

```
MultiValuedMap<String, String> map = new HashSetValuedHashMap<>();
map.put("key1", "value1");
map.put("key1", "value1");
assertThat((Collection<String>) map.get("key1"))
  .containsExactly("value1");
```

Ambas **implementaciones anteriores no son seguras para subprocesos** .

Veamos cómo podemos utilizar el decorador *UnmodifiableMultiValuedMap* para que sean inmutables:

```
@Test(expected = UnsupportedOperationException.class)
public void givenUnmodifiableMultiValuedMap_whenInserting_thenThrowing
    MultiValuedMap<String, String> map = new ArrayListValuedHashMap<>(
    map.put("key1", "value1");
    map.put("key1", "value2");
    MultiValuedMap<String, String> immutableMap =
      MultiMapUtils.unmodifiableMultiValuedMap(map);
    immutableMap.put("key1", "value3");
}
```

5. Guava *Multimap*

Guava es la API de Google Core Libraries for Java.

El *com.google.common.collect.* ***La*** *(https://google.github.io/guava/releases/23.0/api/docs/com/google/common/collect/Multimap.html)* interfaz ***Multimap*** *(https://google.github.io/guava/releases/23.0/api/docs/com/google/common/collect/Multimap.html)* está ahí desde la versión 2. Al momento de escribir este artículo, el último lanzamiento es el 25, pero desde la versión 23 se ha dividido en diferentes ramas para *jre* y *android* (*25.0-jre* y *25.0-android*), seguiremos usando la versión 23 para nuestros ejemplos.

Comencemos por importar guayaba en nuestro proyecto:

```
<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <version>23.0</version>
</dependency>
```

Guava siguió el camino de las múltiples implementaciones desde el comienzo.

El más común es el *com.google.common.collect.* ***ArrayListMultimap*** *(https://google.github.io/guava/releases/23.0/api/docs/com/google/common/collect/ArrayListMultimap.html)* , que utiliza un *HashMap* respaldado por una *ArrayList* para cada valor:

```
Multimap<String, String> map = ArrayListMultimap.create();
map.put("key1", "value2");
map.put("key1", "value1");
assertThat((Collection<String>) map.get("key1"))
  .containsExactly("value2", "value1");
```

Como siempre, preferimos las implementaciones inmutables de la interfaz Multimap: *com.google.common.collect.* ***ImmutableListMultimap*** *(https://google.github.io/guava/releases/23.0/api/docs/com/google/common/collect/ImmutableListMultimap.html)* y *com.google.common.collect.* ***ImmutableSetMultimap*** *(https://google.github.io/guava/releases/23.0/api/docs/com/google/common/collect/ImmutableSetMultimap.html)* .

5.1. Implementaciones comunes de mapas

Cuando necesitamos una implementación de *Mapa* específica , lo primero que debemos hacer es verificar si existe, porque probablemente la Guayaba ya lo haya implementado.

Por ejemplo, podemos usar *com.google.common.collect.* ***LinkedHashMultimap*** *(https://google.github.io/guava/releases/23.0/api/docs/com/google/common/collect/LinkedHashMultimap.html)* , que conserva el orden de inserción de claves y valores:

```
Multimap<String, String> map = LinkedHashMultimap.create();
map.put("key1", "value3");
map.put("key1", "value1");
map.put("key1", "value2");
assertThat((Collection<String>) map.get("key1"))
  .containsExactly("value3", "value1", "value2");
```

Alternativamente, podemos usar un *com.google.common.collect.* ***TreeMultimap*** *(https://google.github.io/guava/releases/23.0/api/docs/com/google/common/collect/TreeMultimap.html)* , que itera claves y valores en su orden natural:

```
Multimap<String, String> map = TreeMultimap.create();
map.put("key1", "value3");
map.put("key1", "value1");
map.put("key1", "value2");
assertThat((Collection<String>) map.get("key1"))
  .containsExactly("value1", "value2", "value3");
```

5.2. Forjar nuestro *MultiMap* personalizado

Muchas otras implementaciones están disponibles.

Sin embargo, es posible que deseemos decorar un *mapa* y / o una *lista* aún no implementada.

Afortunadamente, la guayaba tiene un método de fábrica que nos permite hacerlo: *Multimap.newMultimap ()* *(https://google.github.io/guava/releases/23.0/api/docs/com/google/common/collect/Multimaps.html#newMultimap-java.util.Map-com.google.common.base.Supplier-)* .

6. Conclusión

Hemos visto cómo almacenar múltiples valores para una clave en un Mapa en todas las principales formas existentes.

Hemos explorado las implementaciones más populares de Apache Commons Collections y Guava, que deberían preferirse a las soluciones personalizadas cuando sea posible.

Como siempre, el código fuente completo está disponible en Github (https://github.com/eugenp/tutorials/blob/master/core-java-collections/src/test/java/com/baeldung/java/map/MapMultipleValuesTest.java) .

Deja una respuesta

¡Sé el primero en comentar!

Start the discussion

Suscribir

CATEGORÍAS

PRIMAVERA (HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
DESCANSO (HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SEGURIDAD (HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCIA (HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/)
HTTPCLIENT (HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTP://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIE

¿Cuál de estos es el más cercano a su trabajo / función actual?

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

TUTORIAL "VOLVER A LO BÁSICO" DE JAVA (HTTP://WWW.BAELDUNG.COM/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (HTTP://WWW.BAELDUNG.COM/JACKSON)
TUTORIAL DE HTTPCLIENT 4 (HTTP://WWW.BAELDUNG.COM/HTTPCLIENT-GUIDE)
REST CON SPRING TUTORIAL (HTTP://WWW.BAELDUNG.COM/REST-WITH-SPRING-SERIES/)
TUTORIAL DE SPRING PERSISTENCE (HTTP://WWW.BAELDUNG.COM/PERSISTENCE-WITH-SPRING-SERIES/)
SEGURIDAD CON SPRING (HTTP://WWW.BAELDUNG.COM/SECURITY-SPRING)

ACERCA DE

ACERCA DE BAELDUNG (HTTP://WWW.BAELDUNG.COM/ABOUT/)
LOS CURSOS (HTTP://COURSES.BAELDUNG.COM)
TRABAJO DE CONSULTORÍA (HTTP://WWW.BAELDUNG.COM/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
EL ARCHIVO COMPLETO (HTTP://WWW.BAELDUNG.COM/FULL_ARCHIVE)
ESCRIBIR PARA BAELDUNG (HTTP://WWW.BAELDUNG.COM/CONTRIBUTION-GUIDELINES)
CONTACTO (HTTP://WWW.BAELDUNG.COM/CONTACT)
INFORMACIÓN DE LA COMPAÑÍA (HTTP://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO)
TÉRMINOS DE SERVICIO (HTTP://WWW.BAELDUNG.COM/TERMS-OF-SERVICE)
POLÍTICA DE PRIVACIDAD (HTTP://WWW.BAELDUNG.COM/PRIVACY-POLICY)
EDITORES (HTTP://WWW.BAELDUNG.COM/EDITORS)
KIT DE MEDIOS (PDF) (HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-+MEDIA+KIT.PDF)