[java] Obtener un elemento de un conjunto

Question

¿Por qué `Set` no proporciona una operación para obtener un elemento que sea igual a otro elemento?

```
Set<Foo> set = ...;
...
Foo foo = new Foo(1, 2, 3);
Foo bar = set.get(foo);   // get the Foo element
from the Set that equals foo
```

Puedo preguntar si el `Set` contiene un elemento igual a la `bar` , entonces ¿por qué no puedo obtener ese elemento? :(

Para aclarar, el método `equals` se reemplaza, pero solo ve rifica uno de los campos, no todos. Entonces, dos objetos `Foo` que se consideran iguales pueden tener diferentes val ores, por eso no puedo usar `foo` .

Answers

Para responder a la pregunta precisa " **¿** Por **qué** `Set` no proporc iona una operación para obtener un elemento que iguale a otro e lemento?", La respuesta sería: porque los diseñadores del marco de recopilación no estaban muy atentos. No anticiparon su caso d e uso legítimo, trataron ingenuamente de "modelar la abstracción del conjunto matemático" (del javadoc) y simplemente olvidaron agregar el útil método `get()` .

Ahora a la pregunta implícita " **¿cómo** se obtiene el elemento?": C reo que la mejor solución es usar un `Map<E,E>` lugar de un `Set <E>` , para mapear los elementos a sí mismos. De esta forma, pue de recuperar de manera *eficiente* un elemento del "conjunto", ya que el método get () del `Map` encontrará el elemento utilizando u na tabla hash eficiente o un algoritmo de árbol. Si quisiera, podría escribir su propia implementación de `Set` que ofrezca el método `get()` adicional, encapsulando el `Map` .

Las siguientes respuestas son malas o incorrectas en mi opinión:

"No necesita obtener el elemento, porque ya tiene un objeto igua l": la afirmación es incorrecta, como ya mostró en la pregunta. Do s objetos que son iguales todavía pueden tener un estado diferen te que no es relevante para la igualdad del objeto. El objetivo es o btener acceso a este estado del elemento contenido en el `Set` , no al estado del objeto utilizado como "consulta".

"No tienes otra opción más que usar el iterador": es una búsqued a lineal sobre una colección que es totalmente ineficiente para co njuntos grandes (irónicamente, internamente, el `Set` está organi zado como un mapa hash o árbol que se puede consultar de man era eficiente). No lo hagas! He visto problemas severos de rendim iento en sistemas de la vida real al usar ese enfoque. En mi opinió n, lo que es terrible acerca del método `get()` falta no es tanto q ue sea un poco engorroso evitarlo, sino que la mayoría de los pro gramadores usarán el enfoque de búsqueda lineal sin pensar en l as implicaciones.

Lo siguiente puede ser un enfoque

```
 SharedPreferences se_get = getSharedPreferences("points",MODE_PRIVATE);
Set<String> main = se_get.getStringSet("mydata",null);
for(int jk = 0 ; jk < main.size();jk++)
{
   Log.i("data",String.valueOf(main.toArray()[jk]));
}
```

El conjunto predeterminado en Java, desafortunadamente, no está diseñado para proporcionar una operación "get", como jschreine r explicó con precisión.

Las soluciones de usar un iterador para encontrar el elemento de interés (sugerido por dacwe) o para eliminar el elemento y volver a ag regarlo con sus valores actualizados (sugerido por KyleM), podrían funcionar, pero pueden ser muy ineficientes.

Sobrescribir la implementación de iguales para que los objetos no iguales sean "iguales", como afirma correctamente David Ogren , pue de causar fácilmente problemas de mantenimiento.

Y usar un Mapa como un reemplazo explícito (como lo sugirieron muchos), imho, hace que el código sea menos elegante.

Si el objetivo es obtener acceso a la instancia original del elemento contenido en el conjunto (espero haber entendido correctamente su caso de uso), aquí hay otra posible solución.

Personalmente tuve su misma necesidad al desarrollar un videojuego cliente-servidor con Java. En mi caso, cada cliente tenía copias de l os componentes almacenados en el servidor y el problema era siempre que un cliente necesitaba modificar un objeto del servidor.

Pasar un objeto a través de Internet significaba que el cliente tenía diferentes instancias de ese objeto de todos modos. Para hacer coinc idir esta instancia "copiada" con la original, decidí usar UUID de Java.

Así que creé una clase abstracta UniqueItem, que automáticamente otorga una identificación única aleatoria a cada instancia de sus sub clases.

Este UUID se comparte entre el cliente y la instancia del servidor, por lo que de esta manera podría ser fácil combinarlos simplemente us ando un Mapa.

Sin embargo, usar directamente un mapa en un caso de uso similar aún no era elegante. Alguien podría argumentar que usar un Mapa podría ser más complicado de mantener y manejar.

Por estos motivos, implementé una biblioteca llamada MagicSet, que hace que el uso de un mapa sea "transparente" para el desarrollad or.

https://github.com/ricpacca/magicset

Al igual que el Java HashSet original, un MagicHashSet (que es una de las implementaciones de MagicSet proporcionadas en la bibliotec a) utiliza un HashMap de respaldo, pero en lugar de tener elementos como claves y un valor ficticio como valores, usa el UUID del eleme nto como clave y el elemento en sí mismo como valor. Esto no causa gastos generales en el uso de la memoria en comparación con un HashSet normal.

Además, un MagicSet se puede usar exactamente como un conjunto, pero con algunos métodos más que proporcionan funcionalidades adicionales, como getFromId (), popFromId (), removeFromId (), etc.

El único requisito para usarlo es que cualquier elemento que desee almacenar en un MagicSet necesite extender la clase abstracta Uniq ueItem.

Aquí hay un ejemplo de código, imaginando recuperar la instancia original de una ciudad desde un MagicSet, dada otra instancia de esa ciudad con el mismo UUID (o incluso solo su UUID).

```
 class City extends UniqueItem {

    // Somewhere in this class

    public void doSomething() {
        // Whatever
    }
}

public class GameMap {
    private MagicSet<City> cities;

    public GameMap(Collection<City> cities) {
        cities = new MagicHashSet<>(cities);
    }

    /*
     * cityId is the UUID of the city you want to retrieve.
     * If you have a copied instance of that city, you can simply
     * call copiedCity.getId() and pass the return value to this method.
     */
    public void doSomethingInCity(UUID cityId) {
        City city = cities.getFromId(cityId);
        city.doSomething();
    }

    // Other methods can be called on a MagicSet too
}
```

Si tiene un objeto igual, ¿por qué necesita uno del conjunto? Si es "igual" solo por una clave, un `Map` sería una mejor opción.

De todos modos, lo siguiente lo hará:

```
Foo getEqual(Foo sample, Set<Foo> all) {
  for (Foo one : all) {
    if (one.equals(sample)) {
      return one;
    }
  }
  return null;
}
```

Con Java 8 esto puede convertirse en un trazador de líneas:

```
return all.stream().filter(sample::equals).findAny().orElse(null);
```

Sé que esto se ha preguntado y respondido hace mucho tiempo; sin embargo, si alguien está interesado, esta es mi solución: clase de conjunto personalizado respaldado por HashMap:

http://pastebin.com/Qv6S91n9

Puede implementar fácilmente todos los demás métodos de Conjunto.

He estado allí hecho eso! Si está usando guayaba, una forma rápida de convertirlo en un mapa es:

```
Map<Integer,Foo> map = Maps.uniqueIndex(fooSet, Foo::getKey);
```

Porque cualquier implementación particular de Set puede o no ser de acceso aleatorio .

Siempre puede obtener un iterator y recorrer el Conjunto, utilizando el método `next()` los iteradores para devolver el resultado que desea una vez que encuentre el elemento igual. Esto funciona independientemente de la implementación. Si la implementación NO es de acceso aleatorio (imagine un conjunto respaldado con una lista vinculada), un método `get(E element)` en la interfaz sería engañoso, ya que tendría que iterar la colección para encontrar el elemento a devolver, y un `get(E element)` parece implicar que esto sería necesario, que el Conjunto podría saltar directamente al elemento para obtener.

`contains()` puede o no tener que hacer lo mismo, por supuesto, dependiendo de la implementación, pero el nombre no parece prestarse al mismo tipo de malentendidos.

Con Java 8 puedes hacer:

```
Foo foo = set.stream().filter(item->item.equals(theItemYouAreLookingFor)).findFirst().get();
```

Pero tenga cuidado, .get () arroja una NoSuchElementException, o puede manipular un elemento opcional.

Por qué:

Parece que Set juega un papel útil al proporcionar un medio de comparación. Está diseñado para no almacenar elementos duplicados.

Debido a esta intención / diseño, si se tuviera que obtener () una referencia al objeto almacenado, luego mutarlo, es posible que las intenciones de diseño de Set pudieran frustrarse y provocar un comportamiento inesperado.

De los JavaDocs

> Se debe tener mucho cuidado si los objetos mutables se usan como elementos establecidos. El comportamiento de un conjunto no se especifica si el valor de un objeto se cambia de una manera que afecta a las comparaciones iguales mientras que el objeto es un elemento en el conjunto.

Cómo:

Ahora que se han introducido Streams uno puede hacer lo siguiente

```
mySet.stream()
.filter(object -> object.property.equals(myProperty))
.findFirst().get();
```


Share

Links

- Crear ArrayList desde array
- C # Establecer colección?
- ¿Cómo puedo llamar a un constructor de otro en Java?
- El método tiene el mismo borrado que otro método en tipo
- Cómo convertir una matriz a un conjunto en Java
- Gson: Cómo excluir campos específicos de la serialización sin anotaciones
- Formas de iterar sobre una lista en Java

Tags

java collections set equals

Español Top