(http://baeldung.com)

Cómo encontrar un elemento en una lista con Java

Última modificación: 5 de mayo de 2018

por baeldung (http://www.baeldung.com/author/baeldung/)
(http://www.baeldung.com/author/baeldung/)

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

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

1. Información general

Encontrar un elemento en una lista es una tarea muy común que encontramos como desarrolladores.

En este tutorial rápido, cubriremos diferentes formas en que podemos hacer esto con Java.

2. Configuración

Comencemos definiendo un POJO de *cliente* :

```
public class Customer {

    private int id;
    private String name;

    // getters/setters, custom hashcode/equals
}
```

Y una *ArrayList* (http://www.baeldung.com/java-arraylist) de clientes:

```
List<Customer> customers = new ArrayList<>();
customers.add(new Customer(1, "Jack"));
customers.add(new Customer(2, "James"));
customers.add(new Customer(3, "Kelly"));
```

Tenga en cuenta que hemos redefinido *hashCode* y *es igual* en nuestra clase de *clientes* .

En función de nuestra implementación actual de *iguales* , dos objetos del *Cliente* con el mismo *ID* se considerarán iguales.

Utilizaremos esta lista de *clientes* en el camino.

3. Usando la API de Java

Java en sí proporciona varias formas de encontrar un elemento en una lista:

- **El método *contiene***
- **El método *indexOf***
- **Un bucle for ad-hoc** , y
- **La API *Stream***

3.1. *contiene ()*

La lista expone un método llamado *contiene* :

```
boolean contains(Object element)
```

Como su nombre indica, este método devuelve *verdadero* si la lista contiene el *elemento* especificado , y devuelve *falso* en caso contrario.

Entonces, cuando solo necesitamos verificar si existe un elemento específico en nuestra lista, podemos hacer:

```
Customer james = new Customer(2, "James");
if (customers.contains(james)) {
    // ...
}
```

3.2. *índice de()*

indexOf es otro método útil para encontrar elementos:

```
int indexOf(Object element)
```

Este método devuelve el índice de la primera aparición del *elemento* especificado en la lista dada, o -1 si la lista no contiene el *elemento* .

Entonces, lógicamente, si este método devuelve algo distinto de -1, sabemos que la lista contiene el elemento:

```
if(customers.indexOf(james) != -1) {
    // ...
}
```

La principal ventaja de utilizar este método es que puede indicarnos la posición del elemento especificado en la lista dada.

3.3. Bucle básico

Pero, ¿y si queremos hacer una búsqueda de un elemento basada en el campo? Digamos que estamos anunciando una lotería y tenemos que declarar como ganador a un *cliente* con un *nombre* específico .

Para tales búsquedas basadas en el campo, podemos recurrir a la iteración.

Una forma tradicional de iterar a través de una lista es usar una de las (http://www.baeldung.com/java-loops) construcciones de bucle (http://www.baeldung.com/java-loops) de Java (http://www.baeldung.com/java-loops) . En cada iteración, comparamos el elemento actual en la lista con el elemento que estamos buscando para ver si coincide:

```
public Customer findUsingEnhancedForLoop(
  String name, List<Customer> customers) {

    for (Customer customer : customers) {
        if (customer.getName().equals(name)) {
            return customer;
        }
    }
    return null;
}
```

Aquí, el *nombre se* refiere al nombre que estamos buscando en la lista de *clientes* dada . Este método devuelve el primer objeto del *Cliente* en la lista con un *nombre* coincidente , y *nulo* si no existe dicho *Cliente* .

3.4. Bucle con un *iterador*

Iterador (http://www.baeldung.com/java-iterator) es otra forma de atravesar una lista de artículos.

Simplemente podemos tomar nuestro ejemplo anterior y modificarlo un poco:

```
public Customer findUsingIterator(
  String name, List<Customer> customers) {
    Iterator<Customer> iterator = customers.iterator();
    while (iterator.hasNext()) {
        Customer customer = iterator.next();
        if (customer.getName().equals(name)) {
            return customer;
        }
    }
    return null;
}
```

Y el comportamiento es el mismo que antes.

3.5. Java 8 *Stream* API

A partir de Java 8, también podemos usar *Streams API* (http://www.baeldung.com/java-8-streams) para encontrar un elemento en una *lista.*

Para encontrar un elemento que coincida con los criterios específicos en una lista determinada, nosotros:

- invoke *stream ()* en la lista
- llama al *método f ilter ()* con un *predicado* apropiado
- llamar al constructo *findAny ()* que devuelve **el primer elemento que coincide con el predicado de *filtro* envuelto en un *Opcional*,** si tal elemento existe

```
Customer james = customers.stream()
  .filter(customer -> "James".equals(customer.getName()))
  .findAny()
  .orElse(null);
```

Para mayor comodidad, establecemos por defecto *null* en caso de que un *Opcional* esté vacío, pero esto podría no ser siempre la mejor opción para cada escenario.

4. Bibliotecas de terceros

Ahora, mientras Stream API es más que suficiente, **¿qué deberíamos hacer si estamos atascados en una versión anterior de Java?**

Afortunadamente, hay muchas bibliotecas de terceros como Google Guava y Apache Commons que podemos usar.

4.1. Google Guava

Google Guava proporciona una funcionalidad similar a lo que podemos hacer con las transmisiones:

```
Customer james = Iterables.tryFind(customers,
  new Predicate<Customer>() {
      public boolean apply(Customer customer) {
          return "James".equals(customer.getName());
      }
  }).orNull();
```

Como podemos con *Stream* API, opcionalmente podemos optar por devolver un valor predeterminado en lugar de *null*.

```
Customer james = Iterables.tryFind(customers,
  new Predicate<Customer>() {
      public boolean apply(Customer customer) {
          return "James".equals(customer.getName());
      }
  }).or(customers.get(0));
```

El código anterior seleccionará el primer elemento de la lista si no se encuentra coincidencia.

Y no olvide que Guava arroja una *NullPointerException* si la lista o el predicado es *nulo* .

4.2. Apache Commons

Podemos encontrar un elemento casi de la misma manera usando Apache Commons:

```
Customer james = IterableUtils.find(customers,
  new Predicate<Customer>() {
      public boolean evaluate(Customer customer) {
          return "James".equals(customer.getName());
      }
  });
```

Sin embargo, hay un par de diferencias importantes:

1. Apache Commons simplemente devuelve *null* si pasamos una lista *nula*
2. **No proporciona funcionalidad de valor por defecto como *tryFind* de Guava**

5. Conclusión

En este artículo, hemos aprendido diferentes formas de encontrar un elemento en una *lista,* comenzando con verificaciones rápidas de existencia y finalizando con búsquedas basadas en el campo.

También analizamos las bibliotecas de terceros *Google Guava* y *Apache Commons* como alternativas a Java 8 *Streams* API.

Gracias por visitarnos, y recuerda revisar toda la fuente de estos ejemplos en GitHub. (https://github.com/eugenp/tutorials/tree/master/core-java-8)

Deja una respuesta

¡Sé el primero en comentar!

Start the discussion

Suscribir

CATEGORÍAS

SERIE

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)