Jose Luis Rodríguez Villapecellín

Consultor tecnológico de desarrollo de proyectos informáticos.
Ingeniero Técnico en Informática de Sistemas y master en Ingeniería del Software para la Web.
Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación
Somos expertos en Java/Java EE

Expresiones Lambda con Java 8

diciembre 4, 2015 Jose Luis Rodríguez Villapecellín 10 comentarios Tutoriales 5014 visitas

Entre las múltiples novedades que nos brinda Java 8 encontramos las expresiones lambda.
En este tutorial veremos en que consisten, que tipos existen, como crearlas y como utilizarlas.

Índice de contenidos

Los más visitados de la semana

Introducción a NestJS
VirtualBox. Configuración de la conexión de red.
Parte 1. Aprendiendo HTML para crear una página web
Page Analytics
Instalación y uso del plugin de comentarios de Facebook en n...

1. Introducción

Las expresiones lambda son funciones anónimas, es decir, funciones que no necesitan una clase.
su sintáxis básica se detalla a continuación:

(parámetros) -> { cuerpo-lambda }

1. El operador lambda (->) separa la declaración de parámetros de la declaración del cuerpo de la función.

3. Cuerpo de lambda:
   - Cuando el cuerpo de la expresión lambda tiene una única línea no es necesario utilizar las llaves y no necesitan especificar la clausula return en el caso de que deban devolver valores.
   - Cuando el cuerpo de la expresión lambda tiene más de una línea se hace necesario utilizar las llaves y es necesario incluir la clausula return en el caso de que la función deba devolver un valor .

Algunos ejemplos de expresiones lambda pueden ser:

- z -> z + 2
- () -> System.out.println(" Mensaje 1 ")
- (int longitud, int altura) -> { return altura * longitud; }
- (String x) -> { String retorno = x;
  retorno = retorno.concat(" ***");
  return retorno; }

Como hemos visto las expresiones lambda son funciones anónimas y pueden ser utilizadas allá donde el tipo aceptado sea una interfaz funcional pero… ¿qué es una interfaz funcional?
Una interfaz funcional es una interfaz con uno y solo un método abstracto. La declaración es exactamente igual que las interfaces normales con dos características adicionales:

- Tiene un único método abstracto, como ya hemos dicho.
- De manera opcional puede estar anotada como `@FunctionalInterface` .

El motivo de que la interfaz tenga un único método abstracto es que será la expresión lambda la que proveerá de la implementación para dicho método.

A continuación algunos ejemplos de interfaz funcional:

```
@FunctionalInterface
public interface Runnable {
    public abstract void run();
}
```

```
public interface MiInterfaz {
    default void saluda() {
        System.out.println("Un saludo!");
    }
    public abstract int calcula(int dato1, int dato2);
}
```

```
@FunctionalInterface
public interface Comparator {
    // Se eluden los métodos default y estáticos
    int compare(T o1, T o2);
    // El método equals(Object obj) es implicitamente implementado por la clase objeto.
    boolean equals(Object obj);
}
```

2. Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware: MacBook Pro 17′ (2.66 GHz Intel Core i7, 8GB DDR3 SDRAM).
- Sistema Operativo: Mac OS X Lion 10.10.3.
- NVIDIA GeForce GT 330M 512Mb.
- Crucial MX100 SSD 512 Gb.

3. Tipos

Las expresiones lambda puede clasificarse de la siguiente manera:

- Consumidores.
- Proveedores.
- Funciones

A continuación iremos detallando uno a uno y contando en que consisten.

3.1. Consumidores

Se trata de aquellas expresiones lambda que aceptan un solo valor y no devuelven valor alguno.

```
1 String message -> System.out.println(message);
```

Las expresiones BiConsumidoras, un caso especial de las expresiones consumidoras, son aquellas que toman dos valores como parámetro y no devuelven resultado.

```
1 (String key, String value) -> System.out.println("Key: %s, value: %s%n", key, value);
```

3.2. Proveedores

En este caso se trata de expresiones que no tienen parámetros pero devuelven un resultado.

```
1 () -> return createRandomInteger()
```

3.3. Funciones

Aquellas expresiones que aceptan un argumento y devuelven un valor como resultado y cuyos tipos no tienen porque ser iguales.

```
1 Order persistedOrder -> persistedOrder.getIdientifier();
```

Las BiFunciones son aquellas expresiones de tipo función que aceptan dos argumentos y devuelven un resultado.

```
1 (Address address, String name) -> new Person(name, address);
```

3.3.1 Operadores Unarios

Caso especial de funciones en las que tanto el parámetro como el valor devuelto son del mismo tipo.

```
1 String message -> message.toLowerCase()
```

3.3.2 Operadores Binarios

Igual que en el caso de los Operadores Unarios, se trata de un caso especial de funciones en las que los dos argumentos y el resultado son del mismo tipo.

```
1 (String message, String anotherMesssage) -> message.concat(anotherMessage);
```

3.4. Predicados

Se trata de expresiones que aceptan un parámetro y devuelven un valor lógico.

```
1 String message -> message.length > 50
```

Como en los casos anteriores, se pueden tener BiPredicados, predicados que en lugar de tener un parámetro, tienen dos.

```
1 (path, attr) -> String.valueOf(path).endsWith(".js") && attr.size() > 1024
```

4. Referencias a métodos

Las referencias a los métodos nos permiten reutilizar un método como expresión lambda. Para hacer uso de las referencias a métodos basta con utilizar la siguiente sintáxis: `referenciaObjetivo::nombreDelMetodo`.

```
File::canRead // en lugar de File f -> f.canRead();
```

Con las referencias a los métodos se ofrece una anotación más rápida para expresiones lambda simples y

- Métodos de instancia de un tipo.
- Métodos de instancia de un objeto existente.

Ejemplo de uso con método estático:

```
(String info) -> System.out.println(info) // Expresión lambda sin referencias.
System.out::println // Expresión lambda con referencia a método estático.
```

Ejemplo de uso con método de un tipo:

```
(Student student, int registryIndex) -> student.getRegistry(registryIndex) // Expresión lambda sin
 referencias.
Student::getRegistry // Expresión lambda con referencia a método de un tipo.
```

Ejemplo de uso con método de un objeto existente:

```
Student student -> getMarks(student) // Expresión lambda sin referencias.
this::getMarks // Expresión lambda con referencia a método de un objeto existente.
```

5. Uso

Como hemos visto con anterioridad, las lambdas pueden ser utilizadas allá donde el tipo de parámetros aceptados sea una interfaz funcional.
Este es el caso de muchas de las funcionalidades que ofrece `java.util.stream`, nuevo Api que aparece con Java 8, que permite la programación funcional sobre un flujo de valores sin estructura.

Veamos algunos ejemplos:

Ejemplo 1:

```
List aList = ...
...
aList.stream()
     .findFirst(element -> element.getValue() == VALUE_TO_COMPARE)
     .ifPresent(System.out::println);
```

Mediante el codigo anterior hacemos uso de lambdas para:

- `element -> element.getValue == VALUE_TO_COMPARE` : otorgamos un predicado a la función findFirst de la API Stream, está función utilizará el predicado para devolver un Optional (otra de las novedades de Java 8 que ya trató Daniel Díaz en su tutorial Jugando con la clase Optional en Java 8.) con el primer valor que corresponda la expresión lambda.
- `System.out::println` : establecemos un consumidor para la función ifPresent, de la clase Optional, que en caso de existir el valor, ejecutará la expresión lambda.

Ejemplo 2:

```
Map<String, Integer> map = new TreeMap<>();
map.put....
...
StringBuilder stringBuilder = new StringBuilder();
map.forEach((letter, number) -> stringBuilder.append(letter.concat(String.valueOf(number))));
System.out.println(stringBuilder.toString());
```

En el anterior código utilizamos una lambda para procesar valores, a través de un consumidor, que concatena los valores enteros que se obtienen del mapa.

Ejemplo 3:

```
try (BufferedReader reader = Files.newBufferedReader(Paths.get("SomeLines.txt"), StandardCharsets.U
TF_8)) {
    reader.lines()
          .flatMap(line -> Stream.of(line.split(WORD_REGEXP)))
          .distinct()
          .map(String::toLowerCase)
          .forEach(System.out::println);
}
```


- `line -> Stream.of(line.split(WORD_REGEXP))` : Devuelve un Stream con los valores obtenidos de aplicar la división de la línea mediante la expresión regular. El método flatMap irá recogiendo los streams generados de esta manera y los convertirá a un único stream que podrá ser procesado con posterioridad.
- `String::toLowerCase` : transforma los strings del stream a minúsculas.
- `System.out::println` : ya lo vimos con anterioridad, muestra por consola cada string.

6. Conclusiones

Las expresiones lambda de Java 8 nos ofrecen varias mejoras con respecto a las versiones anteriores:

- Nos acerca a la programación funcional.
- Hace nuestro código más preciso y legible, mejorando, en consecuencia, su mantenibilidad.
- Su utilización junto con la API Stream hace más fácil la ejecución concurrente de tareas.

En este tutorial hemos mostrado en qué consisten y qué tipos hay, cuándo pueden utilizarse y las diferentes maneras de utilizarlas. Pendiente de posteriores tutoriales queda tratar aspectos más avanzados como streams finitos e infinitos, utilización de colectores o debugging de streams y expresiones lambda.

7. Referencias

- JDK 8 Mooc: Lambdas and Streams Introduction
- Oracle Lambda Expressions Documentation

Si quieres profundizar en esta y en el resto de novedades que incluye Java 8, recuerda que ahora puedes asistir a los cursos públicos de "Novedades de Java 8", donde te llevaremos de la mano para experimentar y aprender a sacar todo el partido de todos los cambios del lenguaje. Así estarás al día. Puedes comprobar las próximas convocatorias del curso "Novedades Java 8"

expresiones lambda | java 8 | lambda | lambda expressions

Entradas relacionadas

Respuestas (10)

Google Guava: Manejando Colecciones Con Programación Funcional En Java6 Y Java7 | Adictosaltrabajo
abril 4, 2016 at 8:01 am · Responder

[…] quieres saber más sobre Java8 te recomiendo que te consultes este tutorial sobre lambdas en Java8, o mucho mejor, te apuntes a nuestro curso de Java […]

cesar
mayo 29, 2016 at 8:54 pm · Responder

Nunca tan mal explicado… recomiendo ver la explicacion de Oracle y esta en castellano

José Luis Rodriguez Villapecellin
mayo 30, 2016 at 8:43 am · Responder

César, gracias por tomarte tu tiempo en leer y comentar el tutorial, siento que no haya sido de tu gusto.

Siempre que me encuentro críticas constructivas intento mejorar, ¿algo que hayas echado en falta o explicado de manera distinta?

Angel
junio 14, 2016 at 11:42 am · Responder

Pues a mi me parece correcta la explicación y sobre todo es que me ha ayudado.
Está claro que el viento no sopla igual para todos.

José Luis Rodriguez Villapecellin
junio 15, 2016 at 7:09 pm · Responder

Muchas gracias por leer el tutorial Ángel. Me alegro de que te haya sido de ayuda.

Lambdas, Stream y Referencias a métodos | adictosaltrabajo
junio 23, 2016 at 8:30 am · Responder

[…] Expresiones Lambda con Java 8 […]

Benchmarking Java 8: ¿Qué es más rápido? | adictosaltrabajo
junio 24, 2016 at 9:02 am · Responder

[…] método testWith nos permitirá pasar por parámetro el método en forma de Lambda (Proveedor) que queremos realizar para aplicar una serie de acciones antes y después de ejecutar el test, […]

Javi
septiembre 12, 2016 at 10:43 am · Responder

Uff la explicacion teorica esta bastante bien, pero los ejemplos podían ser mejores. He tenido que estar buscando ejemplos prácticos en otros lados, porque los de aquí como explicación valen de poco. Pero gracias por tu articulo.

Spark Framework | adictosaltrabajo
abril 27, 2017 at 4:06 pm · Responder

[…] Si no tienes conocimiento sobre Lambdas te recomiendo que primero eches un ojo al siguiente tutorial: Expresiones Lambda con Java 8 […]

Andrea
enero 9, 2018 at 1:36 am · Responder

La explicación fue sencilla y super entendible :). Me hubiera gustado ver ejemplos un poquito mas complejos, pero en general bien

Name (required) E-Mail (required) Website

Enviar comentario

Lo último

- Arquitectura Serverless con Lambdas sobre AWS
- Terraform para provisionar en AWS
- Introducción a NestJS
- Cómo eliminar el chroma de una fotografía en menos de 5 minutos
- Property-based testing con ScalaCheck.

Datos de contacto

Edificio BestPoint Avd. de Castilla, 1, Planta 2, Oficina 21B (San Fernando de Henares)
Phone: 916 75 33 06
E-Mail: adictos@adictosaltrabajo.com
Web: https://www.autentia.com

Powered by

autentia
Soporte a desarrollo informático