Java 8 - Lambda expression

1. Giới thiệu Lambda

Lamda là một hàm không có tên (unamed function) với các tham số (parameters) và nội dung thực thi (body). Nội dung thực thi của Lamda expression có thể là 1 khối lệnh hoặc 1 biểu thức. Dấu “->” tách biệt các tham số và nội dung thực thi.

Lambda là một trong những thay đổi lớn nhất và được mong đợi nhất của ngôn ngữ Java trong phiên bản phát hành này. Nó cho phép chúng ta xử lý chức năng (functionality) như một đối của phương thức (thông qua function around), hoặc xử lý code như dữ liệu (data): những khái niệm mà mỗi nhà phát triển chức năng rất quen thuộc.

Có rất nhiều ngôn ngữ trên nền tảng JVM (Groovy, Scala…) đã có lambdas từ những ngày đầu tiên, tuy nhiên Java developer thi không có sự lựa chọn cho lambdas với anonymous classes.

Việc thảo luận và thiết kế Lambdas được thực hiện với rất nhiều thời gian và nỗ lực của cộng đồng. Cuối cùng, bản thương mại đã được đề ra, một cấu trúc ngôn ngữ ngắn gọn và súc tích. Trong dạng đơn giản nhất, lambda có thể được biểu diễn như một danh sách các dấu phẩy của các thông số.

2. Cách khai báo và ví dụ

Ví dụ:

```
Arrays.asList( "a", "b", "d" ).forEach( e -> System.out.println( e ) );
```

Chú ý rằng, kiểu đối số e được suy ra từ trình biên dịch compiler. Ngoài ra, bạn cũng có thể cung cấp kiểu đối số một cách tường minh, trong dấu ngoặc đơn.

Ví dụ:

```
Arrays.asList( "a", "b", "d" ).forEach( ( String e ) -> System.out.println( e ) );
```

Trong trường hợp body của lambda phức tạp hơn, nó có thể được đặt trong dấu ngoặc nhọn – như một hàm thông thường trong Java.

Ví dụ:

```
Arrays.asList( "a", "b", "d" ).forEach( e -> {
    System.out.print( e );
    System.out.print( e );
```

```
} );
```

Lambda có thể reference tới thành viên của lớp và các biến cục bộ (chắc chắn rằng các biến này là kiểu final)

Ví dụ: Hai đoạn mã sau là tương đương

```
String separator = ",";
Arrays.asList( "a", "b", "d" ).forEach(
( String e ) -> System.out.print( e + separator )
);
```

và:

```
final String separator = ",";
Arrays.asList( "a", "b", "d" ).forEach(
( String e ) -> System.out.print( e + separator )
);
```

Lambda có thể trả về kết quả giá trị. Kiểu của return value sẽ được suy ra từ trình biên dịch compiler. Câu lệnh return không yêu cầu bắt buộc nếu lambda body chỉ chứa 1 dòng code.

Ví dụ: Hai đoạn mã sau là tương đương

```
Arrays.asList( "a", "b", "d" ).sort( ( e1, e2 ) -> e1.compareTo( e2 ) );
```

và

```
Arrays.asList( "a", "b", "d" ).sort( ( e1, e2 ) -> {
int result = e1.compareTo( e2 );
return result;
} );
```

3. Biểu thức lambda minh bạch và không minh bạch (Explicit and implicit lambda expression)

Mô tả

- Một biểu thức lambda không khai báo kiểu (type) của các tham số (parameter) của nó được gọi là biểu thức lambda không minh bạch (implicit)
- Một biểu thức lambda minh bạch (explicit) là biểu thức có khai báo kiểu của các tham số của nó một cách rõ ràng.
  Trình biên dịch sẽ suy ra các kiểu tham số đối với biểu thức lambda không minh bạch.

Ví dụ 1:

Đoạn code sau tạo một interface với một phương thức duy nhất và sử dụng nó như một loại biểu thức lambda. Khi tạo ra biểu thức lambda chúng ta khai báo tham số s1 có kiểu là Integer

```
public class Main {
    public static void main(String[] args) {
        MyIntegerCalculator myIntegerCalculator = (Integer s1) -> s1 * 2;
        System.out.println("Result: " + myIntegerCalculator.calcIt(5));
    }
}
interface MyIntegerCalculator {
    public Integer calcIt(Integer s1);
}
```

Đoạn code trên sẽ tạo ra kết quả: Result: 10

Ví dụ 2:

Dưới đây là bản demo mà không sử dụng khai báo kiểu cho các tham số. Khi bỏ qua các kiểu, trình biên dịch sẽ tự nhận ra nó.

```
public class Main {
    public static void main(String[] args) {
        MyIntegerCalculator myIntegerCalculator = (s1) -> s1 * 2;
        System.out.println("Result: " + myIntegerCalculator.calcIt(5));
    }
}
interface MyIntegerCalculator {
    public Integer calcIt(Integer s1);
}
```

Đoạn code trên sẽ tạo ra kết quả: Result: 10

4. Tại sao nên sử dụng Lambda

- Giảm số dòng code

Một trong những tiện ích dễ thấy nhất khi sử dụng Lambda Expression là số lượng dòng code được giảm. Chúng ta có thể dễ dàng tạo ra một thể hiện của một functional interface bằng cách sử dụng Lambda Expression hơn là sử dụng một anonymous class.

- Hỗ trợ thực hiện tuần tự (sequential) và song song (parallel)

Một tiện ích khác của Lambda là việc hưởng lợi từ sự hỗ trợ của Stream API cho các tiến trình tuần tự và song song.

Để cho dễ hiểu, chúng ta cùng làm 1 ví dụ đơn giản, chúng ta sẽ viết một phương thức để kiểm tra xem số nhập vào có phải là số nguyên tố hay không.

Theo cách thông thường, ta sẽ viết code như sau, có thể đoạn code chưa tối ưu nhưng thể hiện được điều mình muốn làm:

```
private static boolean isPrime(int number) {
    if(number < 2) return false;
    for(int i=2; i<number; i++){
        if(number % i == 0) return false;
    }
    return true;
}
```

Vấn đề ở đoạn code trên là nó sẽ chạy tuần tự một cách tự nhiên và nếu chúng ta có 1 số cực lớn thì chúng ta sẽ mất nhiều thời gian. Ngoài ra trong đoạn code này có nhiều điểm trả về mà nó sẽ không chạy qua.

Bây giờ chúng ta sẽ xem Lambda giải quyết vấn đề này như thế nào.

```
private static boolean isPrime(int number) {
    return number > 1
        && IntStream.range(2, number).noneMatch(
            index -> number % index == 0);
}
```

IntStream là một chuỗi các phần tử có giá trị kiểu int hỗ trợ các tiến trình tổng hợp tuần tự và song song.

Để dễ đọc hơn, chúng ta cũng có thể viết theo cách sau:

```
private static boolean isPrime(int number) {
    IntPredicate isDivisible = index -> number % index == 0;
    return number > 1
            && IntStream.range(2, number).noneMatch(
                    isDivisible);
}
```

- Truyền hành động (behavior) vào phương thức

Bây giờ chúng ta sẽ truyền hành động vào một phương thức thông qua một ví dụ đơn giản.

```
public static int sumWithCondition(List numbers, Predicate predicate) {
    return numbers.parallelStream()
        .filter(predicate)
        .mapToInt(i -> i)
        .sum();
}
```

Cách sử dụng:

```
//Tổng các số
sumWithCondition(numbers, n -> true)
//Tổng các số chẵn
sumWithCondition(numbers, i -> i%2==0)
//Tổng các số lớn hơn 5
sumWithCondition(numbers, i -> i>5)
```

- Lười biếng nhưng hiệu quả

Một lợi thế của việc sử dụng Lambda là sự lười biếng. Ví dụ, ta sẽ viết một phương thức tìm ra số lẻ lớn nhất trong dãy từ 3 đến 11 và trả về bình phương của nó.

Thông thường, đoạn code sẽ được viết như sau:

```
private static int findSquareOfMaxOdd(List numbers) {
    int max = 0;
    for (int i : numbers) {
        if (i % 2 != 0 && i > 3 && i < 11 && i > max) {
            max = i;
        }
    }
    return max * max;
}
```

Chương trình trên sẽ luôn chạy tuần tự nhưng chúng ta có thể sử dụng Stream API để đạt được điều này với cách lười biếng nhất. Hãy xem cách chúng ta viết lại khi sử dụng Stream API và Lambda Expression.

```
public static int findSquareOfMaxOdd(List numbers) {
    return numbers.stream()
            .filter(NumberTest::isOdd)
            .filter(NumberTest::isGreaterThan3)
```

```
            .filter(NumberTest::isLessThan11)
            .max(Comparator.naturalOrder())
            .map(i -> i * i)
            .get();
}
public static boolean isOdd(int i) {
    return i % 2 != 0;
}
public static boolean isGreaterThan3(int i){
    return i > 3;
}
public static boolean isLessThan11(int i){
    return i < 11;
}
```

Revision #1
Created 29 September 2019 16:07:46 by Laptrinh.vn
Updated 19 October 2019 16:55:59 by Laptrinh.vn