Java 8 - Làm quen nhanh với Lambda Expressions

07/03/2015 / Bởi Nhữ Đình Thuận / trong Java

Tài liệu tiếng Việt về Java 8 Lambda Expressions - một trong những điểm mới qua trọng về mặt ngôn ngữ của Java 8. Tài liệu biên soạn cho chương trình đào tạo Java căn bản tại TechMaster.vn

Chúng ta bắt đầu từ lớp vô danh (Anonymous Class)

```
new ImplementInterface {
 @Override
 public returnType defineMethod(args) {
  body code
 }
}
```

sang tư duy của biểu thức - Lambda Expresions

(args) -> { body code }

Đó là điểm mới về mặt ngôn ngữ trong Java 8.

Một kỹ thuật lập trình trong Java là việc đẩy hành vi (behaviours) vào phương thức (method) bằng việc sử dụng Anonymous class - lớp vô danh. Ví dụ, chúng ta thường cài đặt các listener cho các Swing component như sau:

```
button.addActionListener(new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e) {
   System.out.println("hello");
  }
 });
```

Sang Java 8, chúng ta có một lựa chọn khác là Lambda.

```
button.addActionListener(e -> System.out.println("hello"));
```

rõ ràng cách viết code thứ 2 ngắn gọn, đơn giản và rõ ràng hơn nhiều so với cách thứ nhất. Từ Lambda Expressions, chúng ta sẽ có được những cách thức giải quyết các bài toán trên tập (collection) dễ dàng hơn rất nhiều như lọc, sắp

xếp, duyệt tuần tự, tách,... Ngoài ra, Lambda cũng giúp chúng ta cả tiến hiệu năng (performance) của concurrency features bằng xử lý song song (parallel) trong môi trường đa nhân (multi-processor CPUs). Nội dung dưới đây lần lượt làm quen với Lambda Expression từ đó xây dựng tư duy lập trình chức năng bằng các Anonymous Method.

Như mọi bài viết (article) khác về Lambda, đầu tiên chúng ta làm quen với Runnable interface.

Trước Java 8, chúng ta có thể viết

```
Runnable runnable = new Runnable() {
  @Override
  public void run() { body code }
};
```

Sang đến Java 8, chúng ta có thể viết

```
Runnable runnable = () -> { body code };
```

Khi đã có runnable, chúng ta có thể bắt đầu với thread bằng dòng code.

```
new Thread(runnable).start();
```

Tuy nhiên, cũng có một cách khác bằng việc tạo trực tiếp runnable khi triệu gọi constructor.

```
new Thread(-> { body code }.start();
```

Trong việc cài đặt các listener cho các User Interface component, chúng ta thường viết.

```
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
      body code
    }
});
```

Nhưng một cách ngắn gọn hơn với Lambda.

```
button.addActionListener(e -> body);
```

Một câu hỏi đặt ra là trong trường hợp listener có design nhiều method thì expression sẽ thay thế cho method nào???

Ví dụ, khác với ActionListener, WindowListener có tới 7 methods (tra API) và chúng ta chỉ muốn cài code cho 1 trong 7 methods đó, khi đó có sử dụng lambda được không?

Câu trả lời là có, tuy nhiên hơi phức tạp một chút bằng việc sử dụng default methods. Như chúng ta đa biết, interface là design tăng cường tính đa diện cho đối tượng. Interface quyết định hành vi cho đối tượng trong OOP. Tuy nhiên, trường hợp interface vạch ra quá nhiều hành vi mà lập trình viên chỉ muốn cài đặt một hành vi thì những phương thức khác phải khai báo - cài đặt rỗng dẫn đến code trở nên dài dòng, thừa thãi. Adapter class ra đời như một giải pháp, sang đến Java 8, chúng ta
lại có một phương án nữa là default method.

Xem xét ví dụ sau:

```
public class DemoDefaultMethod {
   public interface Test {
     public void setup();
     public default void run () {
       System.out.println("Hello Tester");
     }
   }

   public static void main(String[] args) {
     Test test = new Test() {
       @Override
       public void setup() {
         System.out.println("Setup environment in here");
       }
     };
     test.run();
   }
 }
```

Interface Test có một default method là run method trong đó không cần thiết lúc nào chúng ta cũng phải cài đặt run method cho các implement từ Test interface.

Như vậy việc có thêm default method thì interface na ná như abstract class trong design.

Một chút thay đổi trong hàm main

```
public static void main(String[] args) {
   Test test = () -> {
     System.out.println("Setup environment in here");
   };
   test.run();
 }
```

Rõ ràng interface Test có 2 method nhưng chỉ cần cài đặt ít nhất 1 method chứ không cần bắt buộc cài đặt cả 2. Ta có thể áp cách thức này trong việc sử dụng listener, chẳng hạn MouseListener. Chúng ta extends từ MouseListener nhưng bỏ ngỏ 1 method là mouseClicked, các method còn lại cài đặt default.

```
interface ClickedListener extends MouseListener {
   @Override
   public default void mouseEntered(MouseEvent e) {}
   @Override
   public default void mouseExited(MouseEvent e) {}
   @Override
   public default void mousePressed(MouseEvent e) {}
   @Override
   public default void mouseReleased(MouseEvent e) {}
 }
```

Và sử dụng

```
button.addMouseListener((ClickedListener)(e)->System.out.println("Clicked !"));
```

Lambda được sử dụng cho việc cài đặt nốt method còn thiếu bắt sự kiện click chuột.

Xem xét việc sử dụng trong sắp xếp với Comparator.

```
 Integer [] values = ...;
 Arrays.sort(values, new Comparator() {
   public int compare(Integer o1, Integer o2) {
     return o2 - o1;
   }
 });
```

Có thể thay thế bằng Lambda.

```
 Arrays.sort(values, (Integer o1, Integer o2) -> {
     return o2 - o1;
   }
 );
```

Bây giờ chúng ta làm quen với Stream design. Xem xét một cách thức viết code truyền thống.

```
List names = Arrays.asList("Nguyen Thi A", "Tran Thi B", "Le Thi C");
 for(String name : names) {
   System.out.println(name);
 }
```

Chúng ta chuyển từ for each cũ sang mô hình mới.

```
names.forEach((name) -> System.out.println(name));
```

Bây giờ ta làm quen với Stream interface.

```
names.stream().forEach((name) -> System.out.println(name));
```

Stream là một design mới từ Java 8 cho phép xử lý một tập như với một luồng I/O trong có bổ sung nhiều thiết kế cho các bài toán đặc thù như lọc, sắp xếp tách tập con, bỏ trùng lặp,... Ngoài ra Stream hỗ trợ cả xử lý song song. Chi tiết, bạn có thể tra Java API.

Chúng ta làm quen util function khi kết hợp với stream.

Bài toán lọc với Predicate

```
Predicate predicate = (String name) -> {
   return name.indexOf("Tran") > -1;
 };
```

```
Stream stream = names.stream().filter(predicate);
 stream.forEach((name) -> System.out.println(name));
```

Create data với Supplier

```
Supplier supplier = () -> { return (int) (Math.random()*1000); };
  for(int i = 0; i < 10; i++) {
    System.out.println(supplier.get());
  }
```

Viết một chức năng với đầu vào cho Consumer

```
  Consumer c = (Integer x) -> { System.out.println(x + 100); };
  c.accept(123);
```

hoặc

```
Consumer c = (String name) -> { System.out.println(name); };
 names.forEach(c);
```

Chuyển đổi với Function.

```
  Function toNumber = (String value) -> {
    return Integer.parseInt(value);
  };
  System.out.println(toNumber.apply("123") + 10);
```

Giả dụ trong bài toán chuyển đổi sang số, nếu người dùng nhập vào 1 giá trị không phải là định dạng số, sẽ có exception xảy ra trong lúc chạy. Ví dụ dưới đây minh họa việc ném ngoại lệ với Lambda.

```
  Function toNumber = (String value) -> {
    try {
      return Integer.parseInt(value);
    } catch(NumberFormatException exp) {
      throw new RuntimeException("Invalid value!");
    }
  };
  System.out.println(toNumber.apply("sa123") + 10);
```

Tài liệu tham khảo:

1. Java 8 Lambda Expressions Tutorial with Examples
2. Java Tutorial
3. Java 8 Lambda Expressions Tutorial
4. 10 Example of Lambda Expressions and Streams in Java 8
5. Java SE 8: Lambda Quick Start
6. Java 8 API

Java và Cái máy giặt
29/05/2015 Techmaster team

BLOG HOME

Nhập môn lập trình Java (Phần 7: Thực thi điều kiện)
08/07/2013 Techmaster team

*Bởi **Nhữ Đình Thuận***

Bỏ ngang ĐH Bách Khoa, tốt nghiệp Aptech, viết phần mềm VietSpider, bán cho khá nhiều khách hàng nước ngoài, làm lập trình viên tại FSoft. Đi dạy ở Aptech, FPT University sau đó dạy Java tại Techmaster. Hiện nay mình đang lead đội Java Core của một công ty tài chính và cố vấn công nghệ Java cho một công ty start up.