**Giới thiệu**

Annotation (Chú thích) được sử dụng để **chú thích** trên một class, một trường (field) hoặc một method để cung cấp hoặc bổ sung các thông tin. Nó hoàn toàn không ảnh hưởng tới code của bạn.

Trong bài có sử dụng các kiến thức:

1. [Optional](https://loda.me/java-8-optional-loda1553606615611)
2. [Functional Interface & Lambda](https://loda.me/java-8-functional-interfaces-and-lambda-expressions-cuc-de-hieu-loda1553649037877)
3. [Java Reflection](https://loda.me/java-huong-dan-java-reflection-loda1554301692892)

Annotation được sử dụng ở 3 dạng:

* Chú thích cho trình biên dịch (Compiler)
* Chú thích cho quá trình build
* Chú thích trong quá trình chạy chương trình (Runtime)

Hẳn bạn đã 1 lần từng thấy cái @Override phải không? nó là một *Annotation chú thích cho trình biên dịch*, để cho trình biên dịch biết hàm đó đã bị ghi đè.

Còn *chú thích cho quá trình build* thì không hẳn có ví dụ cụ thể, nhưng bạn hãy nghĩ tới Maven, Gradle những công cụ build này sẽ có thêm thông tin khi build ứng dụng của bạn khi gặp một số Annotation đặc biệt, và sẽ bổ sung thêm code vào đó.

*Chú thích trong quá trình chạy chương trình* sẽ là nội dung chính của chúng ta hôm nay. Đây là những Annotation mà chỉ khi bạn chạy chương trình rồi thì nó mới tác động tới code. Cùng vào ví dụ để dễ hiểu nhé!

**Khai báo Annotation**

Cách khai báo Annotation là sử dụng @interface

public @interface JsonName {

String value(); // các giá trị trong @interface đều dạng hàm abstract, không tham số

}

vậy là bạn đã có 1 Annotation. Giờ gọi nó ra và sử dụng:

@JsonName(value = "super\_man")

public class SuperMan extends Person {

private String name;

}

Đơn giản phải không? Tuy nhiên, hiện tại Annotation chỉ hiển thị trong code như vậy thôi! chứ nó chả có tác dụng gì cả :))))

Chúng ta cần viết thêm code để xử cái lý cái @JsonName này.

**Khai báo phạm vi cho Annotation**

Chúng ta có thể quy định phạm vi sử dụng của Annotation bằng cách:

@Retention(RetentionPolicy.RUNTIME) // Tồn tại trong lúc chạy chương trình

@Target({ ElementType.TYPE, ElementType.FIELD, ElementType.METHOD}) // Được sử dụng trên class, interface, method, biến

public @interface JsonName {

String value();

}

@Retention: Dùng để chú thích **mức độ tồn tại** của một annotation nào đó. Cụ thể có 3 mức nhận thức tồn tại của vật được chú thích:

1. RetentionPolicy.SOURCE: Tồn tại trên code nguồn, và không được bộ dịch (compiler) nhận ra.
2. RetentionPolicy.CLASS: Mức tồn tại được bộ dịch nhận ra, nhưng không được nhận biết bởi máy ảo tại thời điểm chạy (Runtime).
3. RetentionPolicy.RUNTIME: Mức tồn tại lớn nhất, được bộ dịch (compiler) nhận biết, và máy ảo (jvm) cũng nhận ra khi chạy chương trình.

@Target: Dùng để chú thích **phạm vi sử dụng** của một Annotation

1. ElementType.TYPE - Cho phép chú thích trên Class, interface, enum, annotation.
2. ElementType.FIELD - Cho phép chú thích trường (field), bao gồm cả các hằng số enum.
3. ElementType.METHOD - Cho phép chú thích trên method.
4. ElementType.PARAMETER - Cho phép chú thích trên parameter
5. ElementType.CONSTRUCTOR - Cho phép chú thích trên constructor
6. ElementType.LOCAL\_VARIABLE - Cho phép chú thích trên biến địa phương.
7. ElementType.ANNOTATION\_TYPE - Cho phép chú thích trên Annotation khác
8. ElementType.PACKAGE - Cho phép chú thích trên package.

**Xử lý Annotation**

Bước 1: Chú thích bất kì chỗ nào bạn thích :)))

@JsonName(value = "super\_man")

public class SuperMan {

// Không chú thích, thì chúng ta sẽ coi như lấy tên field là `name` luôn

private String name;

@JsonName("date\_of\_birth")

private LocalDateTime dateOfBirth;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public LocalDateTime getDateOfBirth() {

return dateOfBirth;

}

public void setDateOfBirth(LocalDateTime dateOfBirth) {

this.dateOfBirth = dateOfBirth;

}

}

Bước 2: Viết class xử lý @JsonName:

public class JsonNameProcessor {

public static String toJson(Object object) throws IllegalAccessException {

StringBuilder sb = new StringBuilder(); // Dùng StringBuilder de tao json tu class

Class<?> clazz = object.getClass();

JsonName jsonClassName = clazz.getDeclaredAnnotation(JsonName.class); // Lay ra annotation @JsonName tren Class

sb.append("{\n")

.append("\t\"")

// Lay gia tri cua Annotation, neu annotation la null thi lay ten Class de thay the

.append(Optional.ofNullable(jsonClassName).map(JsonName::value).orElse(clazz.getSimpleName()))

.append("\": {\n"); //

Field fields[] = clazz.getDeclaredFields();

for (int i = 0; i < fields.length; i++) {

fields[i].setAccessible(true); // Set setAccessible = true. De co the truy cap vao private field

JsonName jsonFieldName = fields[i].getDeclaredAnnotation(JsonName.class); // get annotation tren field

sb.append("\t\t\"")

// Lay gia tri cua Annotation, neu annotation la null thi lay ten field thay the

.append(Optional.ofNullable(jsonFieldName).map(JsonName::value).orElse(fields[i].getName())) // L

.append("\": ")

// Neu field la String hoac Object. thi append dau ngoac kep vao

.append(fields[i].getType() == String.class || !fields[i].getType().isPrimitive() ? "\"" : "")

// Lay gia tri cua field

.append(fields[i].get(object))

// Neu field la String hoac Object. thi append dau ngoac kep vao

.append(fields[i].getType() == String.class || !fields[i].getType().isPrimitive()? "\"" : "")

// Nếu là field cuối cùng, thì không append dấu ","

.append(i != fields.length -1 ? ",\n" : "\n");

}

sb.append("\t}\n");

sb.append("}");

return sb.toString();

}

}

Bước 3: Chạy thử:

public static void main(String[] args) throws IllegalAccessException {

SuperMan superMan = new SuperMan(); // Tao doi tuong super man

superMan.setDateOfBirth(LocalDateTime.now());

superMan.setName("loda");

String json =JsonNameProcessor.toJson(superMan);

System.out.println(json);

}

// OUTPUT:

/\*

{

"super\_man": {

"name": "loda",

"date\_of\_birth": "2019-04-03T21:07:23.983"

}

}

\*/