REFLECTION

1. Khái niệm

- Java Refelection là một API trong Java, nó cho phép truy cập các thông tin của class (tên class, fields, methods,…) và chỉnh sửa các field của class (kể cả các field private) trong quá trình run time.
- Ta có thể áp dụng Java Reflection trong những trường hợp không biết object được xử lý là gì (tên class là gì, ở package nào, có những field nào, method nào…).

1. Khái niệm

- Reflection bản chất là đảo ngược quá trình compile source code thành bytecode.
- Tức là ta có thể load ngược class từ bytecode của nó, và có thể kiểm soát các method, fields… của class đó giống như đang làm việc với source code của class đó.

2. Usecase

2.1. Viết một hàm copy object có thể dùng cho mọi object type khác nhau. Thì mình cần phải biết 2 object tham gia có cùng kiểu không, có những field nào, lấy và copy giá trị từng field.

2. Usecase

2.2. Giả sử chúng ta có một list các Animal (interface), các instance của Animal có thể là Dog, Bat, Bird, Cat, Dragon… Mình muốn thực hiện hành động fly() cho tất cả các instance có hỗ trợ hành động này (method fly() không có ở interface Animal). Vấn đề là phải xác định xem instance nào có thể fly().

2. Usecase

2.3. Plugin architecture: giả sử các external plugin dù có chung interface nhưng chúng có những điểm đặc trưng riêng biệt và ta phải phân chia chúng thành các type. Lúc này, lập trình viên sử dụng một static final field trong class để lưu giá trị type đó. Khi application load class (plugin), ta cần đọc field này trong run time để biết type của plugin.

2. Usecase

2.4. Một usecase mà reflection rất hay được sử dụng trong Java đó là thực hiện unit test cùng với sự hỗ trợ của annotation. Ví dụ, thư viện JUnit 4 sẽ sử dụng reflection để xem qua các class để biết các phương thức được gắn annotation @Test và sau đó sẽ gọi chúng khi chạy unit test.

3. Cách sử dụng

3.1 Load Class (chú ý sự khác nhau của từng cách load)

- Biết chính xác class trong compile time

```
Class myObjectClass = MyObject.class;
```

- Lấy được tên class trong run time

```
String className =... //get a string class name
Class class = Class.forName(className);
```

- Lấy class từ một instance của class đó

```
Class class = myInstance.getClass();
```

3. Cách sử dụng

3.2 Constructor

- Get list constructor từ một Class object

```
Class aClass = ... //obtain class object
Constructor[] constructors = aClass.getConstructors();
```

- Get một constructor cụ thể từ đặc điểm của params

```
Constructor constructor =
        aClass.getConstructor(new Class[]{String.class});
```

3. Cách sử dụng

3.2 Constructor

- Create instance từ constructor:

```
Constructor constructor = MyObject.class.getConstructor(String.class);
MyObject myObject = (MyObject)
        constructor.newInstance("constructor-arg1");
```

3. Cách sử dụng

3.3 Field

- Get list field từ một Class object

```
Class aClass = ... //obtain class object
Field[] fields = aClass.getFields();
```

- Get một field với tên cụ thể:

```
Field field = aClass.getField("someField");
```

3. Cách sử dụng

3.3 Field

- Field name and field type:

```
Field field = ... //obtain field object
String fieldName = field.getName();
Object fieldType = field.getType();
```

3. Cách sử dụng

3.3 Field

- Getting and setting field value:

```
Class  aClass = MyObject.class
Field field = aClass.getField("someField");
MyObject objectInstance = new MyObject(); //create new instance
Object value = field.get(objectInstance); //get value of instance by field
field.set(objetInstance, value); //set value of instance with value
```

3. Cách sử dụng

3.4 Method

- Get list method từ một Class object:

```
Class aClass = ... //obtain class object
Method[] methods = aClass.getMethods();
```

- Get một method cụ thể từ tên và danh sách params:

```
Method method =
    aClass.getMethod("doSomething", new Class[]{String.class});
```

3. Cách sử dụng

3.4 Method

- Invoke method using Method object:

```
//get method that takes a String as argument
Method method = MyObject.class.getMethod("doSomething",
String.class);
//first param is null if method is static
Object returnValue = method.invoke(null, "parameter-value1");
```

3. Cách sử dụng

3.5 Private field and method

- Get private field of an object

Class.getDeclaredFields()

Class.getDeclaredField(String name)

3. Cách sử dụng

3.5 Private field and method

- Get private field of an object

```
public class PrivateObject {

   private String privateString = null;

   public PrivateObject(String privateString) {
      this.privateString = privateString;
   }
}
```

3. Cách sử dụng

3.5 Private field and method

- Get private field of an object

```
PrivateObject privateObject = new PrivateObject("The Private Value");

Field privateStringField = PrivateObject.class.
      getDeclaredField("privateString");

privateStringField.setAccessible(true);

String fieldValue = (String) privateStringField.get(privateObject);
System.out.println("fieldValue = " + fieldValue);
```

3. Cách sử dụng

3.5 Private field and method

- Get private method of an object

Class.getDeclaredMethods()

Class.getDeclaredMethod(String name)

3. Cách sử dụng

3.5 Private field and method

- Get private method of an object

```
public class PrivateObject {

   private String privateString = null;

   public PrivateObject(String privateString) {
      this.privateString = privateString;
   }

   private String getPrivateString(){
      return this.privateString;
   }
}
```

3. Cách sử dụng

3.5 Private field and method

- Get private method of an object

```
PrivateObject privateObject = new PrivateObject("The Private Value");

Method privateStringMethod = PrivateObject.class.
      getDeclaredMethod("getPrivateString", null);

privateStringMethod.setAccessible(true);

String returnValue = (String)
      privateStringMethod.invoke(privateObject, null);

System.out.println("returnValue = " + returnValue);
```

4. Demo

2.2. Giả sử chúng ta có một list các Animal (interface), các instance của Animal có thể là Dog, Bat, Bird, Cat, Dragon… Mình muốn thực hiện hành động fly() cho tất cả các instance có hỗ trợ hành động này (method fly() không có ở interface Animal). Vấn đề là phải xác định xem instance nào có thể fly().

Chúng ta sẽ xem code demo giải quyết bài toán này

5. Nhược điểm

Trong trường hợp đã biết rõ cấu trúc class, có quyền truy cập các field, method thì ta không nên sử dụng Java Reflection bởi các lý do sau:

- **Hiệu năng thấp:** đôi khi phải quét classpath để tìm class.
- **Các vấn đề bảo mật:** Việc chỉnh sửa class/object trong quá trình runtime có thể ảnh hưởng tới các thread … khiến cho ứng dụng bị fail.

5. Nhược điểm

- **Khó bảo trì:** Việc Reflection khá khó hiểu với người mới và không dễ để debug, nên sẽ rất khó để có thể tìm ra lỗi. Ngoài ra chúng ta cũng không thể check được một số lỗi trong quá trình compile (không tìm thấy class, không tìm thấy field…)

CẢM ƠN MỌI NGƯỜI ĐÃ LẮNG NGHE

Giờ tới lượt Hưng Trần