Java Refection API

Research

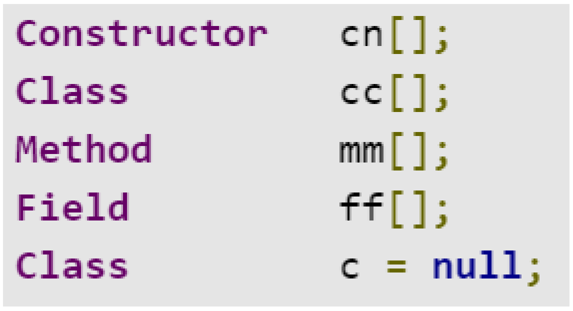
Java Reflection API - what?

1. Grant *dynamic introspection*: the ability to look inside classes that are **already loaded**.
2. The mechanisms used to fetch information about a class.

Consists of 2 components:

1. Classes that represent the various parts of a class file.

   For examples:

   ```
   Constructor    cn[];
   Class          cc[];
   Method         mm[];
   Field          ff[];
   Class          c = null;
   ```

   They are returned by static methods in `java.lang.Class` class (about Class class)

   - `forName` (load a class of given name), `getName`, `newInstance`
   - Added by Reflection API: `getConstructor`, `getConstructors`, `getDeclaredConstructor`, `getMethod`, `getMethods`, `getDeclaredMethods`, `getField`, `getFields`, `getDeclaredFields`, `getSuperclass`, v.v.

2. `java.lang.reflect` class

   > In addition to these methods, many new classes were added to represent the objects that these methods would return. The new classes mostly are part of the `java.lang.reflect` package

Demonstration

Code:

```java
package Demo;
import java.lang.reflect.*;
import java.util.*;
import java.util.Scanner;

public class RefectClass {
    public static void main(String args[]) {
        Constructor cn[];
        Class cc[];
        Method mm[];
        Field ff[]; // ~ all the properties of a class
```

```java
        Class c = null;
        Class supClass;
        String x, y, s1, s2, s3;
        Hashtable classRef = new Hashtable();
        String input = "";

//        System.out.print("Enter a class's name: ");
//        input = new Scanner(System.in).nextLine();
        input = "java.lang.String";

        try {
            c = Class.forName(input);
        } catch (ClassNotFoundException ee) {
            System.out.println("Couldn't find class '" + input + "'");
            System.exit(1);
        }

        x = c.getName();
        y = x.substring(0, x.lastIndexOf("."));
        System.out.println("\nClass name: " + x + "");
        if (y.length() > 0) {
            System.out.println("Package "+y+";");
        }

        System.out.println("\n-------- Fields (properties) of the class -------- ");
        ff = c.getDeclaredFields();
        for(int i = 0; i < ff.length; i++) {
            System.out.println(ff[i].toString());
        }

        System.out.println("\n-------- Constructors of the class ---------");
        cn = c.getDeclaredConstructors();
        for(int i = 0; i < cn.length; i++) {
            System.out.println(cn[i].toString());
        }

        System.out.println("\n---------------------------------------------");
    }
}

```

Results:

```
Class name: java.lang.String
Package java.lang;

-------- Fields (properties) of the class --------
private final char[] java.lang.String.value
private int java.lang.String.hash
private static final long java.lang.String.serialVersionUID
private static final java.io.ObjectStreamField[] java.lang.String.serialPersistentFields
public static final java.util.Comparator java.lang.String.CASE_INSENSITIVE_ORDER

-------- Constructors of the class ---------
public java.lang.String(byte[],int,int)
public java.lang.String(byte[],java.nio.charset.Charset)
public java.lang.String(byte[],java.lang.String) throws java.io.UnsupportedEncodingException
public java.lang.String(byte[],int,int,java.nio.charset.Charset)
public java.lang.String(byte[],int,int,java.lang.String) throws java.io.UnsupportedEncodingException
java.lang.String(char[],boolean)
public java.lang.String(java.lang.StringBuilder)
public java.lang.String(java.lang.StringBuffer)
public java.lang.String(byte[])
public java.lang.String(int[],int,int)
public java.lang.String()
public java.lang.String(char[])
```

Reference:

- https://www.javaworld.com/article/2077015/take-an-in-depth-look-at-the-java-reflection-api.html
- https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html

Dynamic Proxy API

Dynamic proxies

Dynamic proxies allow one single class with one single method to service multiple method calls to arbitrary classes with an arbitrary number of methods.

A dynamic proxy can be thought of as a kind of *Facade*, but one that can pretend to be an implementation of any interface.

Under the cover, **it routes all method invocations to a single handler** – the *invoke()* method.

Tóm lại Dynamic proxies giúp người lập trình backend có thể xem chi tiết và điều khiển việc một object gọi method; override method at runtime.

A working example

Target: Discuss the power of the dynamic proxy by introducing the concept of *views* in your Java programming.

Scenario - problem

> a `Person` class contains the properties `Name`, `Address`, and `PhoneNumber`. Then, there is the `Employee` class, which is a `Person` subclass and contains the additional properties `SSN`, `Department`, and `Salary`. From the `Employee` class, you have the subclass `Manager`, which adds the properties `Title` and one or more `Departments` for which `Manager` is responsible.
>
> Promotion is one idea that you might want to implement in your design. **How would you take a person object and make it an employee object**, and **how would you take an employee object and make it a manager object?** What about the reverse? Also, it might **not be necessary to expose a manager object as anything more than a person object to a particular client**.

Ví dụ một problem:

- Ta có `class A` với method `public String doFunc1()` trả về `"a"`.
- Ta tạo một object từ class A (Instantiate an object) `A objectA = new A()`.
- `ObjectA` vẫn gọi function `doFunc1()` trả về `"a"`, .Nhưng trong lúc run chương trình, dev muốn thay đổi `doFunc1()` cho nó trả về `"b"`
- Vậy phải làm sao? -> Sử dụng Dynamic Proxy.

Flow

1. Tạo một object thuộc class `Proxy` (Dynamic proxy do mình tạo)

```
ViewProxy.newInstance(identity,
        new Class[] { IPerson.class})
```

2. Cast object đó về kiểu mà mình muốn.
3. Mỗi lần gọi hàm, ví dụ `A.method()` của class, ta thấy `Proxy.invoke()` đc gọi ra xử lý. Kết quả của `A.method()` là kết quả return từ `Proxy.invoke()`

Reference

- https://www.baeldung.com/java-dynamic-proxies