The Wayback Machine - https://web.archive.org/web/20210616064616/https://www.arumeinformatica.es/blog/java-reflection-parte-1/

26 de Abril de 2011

Java Reflection (parte 1)

Una de las funcionalidades más potentes y poco conocidas de Java es su soporte para reflexión. Mediante la Java Reflection API (Application Programming Interface) el programador puede inspeccionar y manipular clases e interfaces (así como sus métodos y campos) en tiempo de ejecución, sin conocer a priori (en tiempo de compilación) los tipos y/o nombres de las clases específicas con las que está trabajando.

Quizás pueda parecernos en una primera impresión una funcionalidad con usos limitados. Pero debemos saber que, por ejemplo, muchos frameworks de alto nivel como Hibernate, Spring o Tapestry hacen un uso extensivo de esta API (Application Programming Interface) para facilitarle la vida al programador al permitirle que use simples clases POJO (Plain Old Java Object) para trabajar con ellas. Otros frameworks menos potentes (o versiones antiguas de estos mismos frameworks), obligaban al programador a que sus clases implementaran ciertos interfaces o pertenecieran a complicadas jerarquías de clases, lo cual limitaba la flexibilidad del programador y complicaba la comprensión del código.

Clase

Cuando en nuestro programa queremos usar reflexión para poder trabajar con un objeto del que desconocemos su tipo (en el caso de Java, esto es lo mismo que decir que desconocemos el nombre de su clase), lo primero que debemos hacer es averiguar la clase a la que pertenece. En situaciones normales conoceremos el tipo de un objeto en tiempo de compilación, con lo que obtendríamos su clase de la forma:

```
Class userClass = User.class;
```

En aquellas situaciones en las que no conozcamos el nombre de una clase en tiempo de compilación, podemos obtener su clase en tiempo de ejecución a partir de un `String` que contenga el nombre completo de la clase (incluyendo los paquetes, lo que se conoce como nombre de clase totalmente calificado) usando la función `Class.forName()`:

```
Class userClass = Class.forName("com.test.model.AdminUser");
```

En nuestro ejemplo, la clase es `AdminUser` que pertenece al paquete `com.test.model`.

Si por algún motivo la clase no existiese, se lanzaría una excepción `ClassNotFoundException` para indicarlo.

En sentido inverso, teniendo la clase de un objeto también podremos obtener una cadena con el nombre de la clase:

```
String className = userClass.getName();
```

Este ejemplo nos devuelve el nombre totalmente calificado de la clase. Si queremos sólo el nombre simple (sin el paquete), debemos usar el método `getSimpleName()`.

Información adicional

Podemos obtener el paquete de una clase mediante:

```
Package userPackage = userClass.getPackage();
```

> Nótese que no devuelve un simple `String` con el nombre del paquete, sino un objeto de tipo `Package` que contiene información sobre el paquete y métodos para su manipulación.

También podemos acceder a la superclase de una clase (que será nuevamente un objeto `Class`, de forma que podemos seguir haciendo reflexión sobre él) mediante el método:

```
Class userSuperclass = userClass.getSuperclass();
```

Podemos obtener una lista de los interfaces que implementa una clase (en forma de `array` de objetos `Class`) mediante:

```
Class[] userInterfaces = userClass.getInterfaces();
```

> Debe tenerse en cuenta que sólo se incluyen los interfaces implementados directamente por esta clase, no los que se heredan porque son implementados por alguna superclase de la jerarquía.

Constructores

Podemos acceder a la lista de todos los constructores de una clase mediante el método:

```
Constructor[] userConstructors = userClass.getConstructors();
```

Si conocemos los tipos de los parámetros de un constructor en particular, podemos acceder a ese constructor concreto sin tener que recorrer toda la lista. Esto se hace mediante el método `getConstructor()`, que admite como parámetros un `array` con los tipos específicos de ese constructor (en el mismo orden en el que están declarados). Por ejemplo, si sabemos que nuestra clase de ejemplo contiene un constructor que acepta los parámetros `(String, String, Integer)` podemos acceder a él mediante:

```
Constructor userConstructor = userClass.getConstructor(new Class[] {String.class,
String.class, Integer.class});
```

Si no existiera un constructor con esos parámetros, se lanzaría una excepción de tipo `NoSuchMethodException`.

A la inversa, para un constructor también podemos obtener sus parámetros mediante:

```
Class[] params = userConstructor.getParameterTypes();
```

Una vez hallamos conseguido el constructor deseado, podemos instanciar un objeto mediante el método `newInstance()` con los parámetros adecuados. En nuestro ejemplo:

```
Constructor userConstructor = userClass.getConstructor(new Class[] {String.class,
String.class, Integer.class});
User user = (User) userConstructor.newInstance("Antonio González", "agonzalez@mimail.com",
12);
```

Puedes leer más sobre este tema en nuestro artículo Java Reflection (parte 2).

Creado por Santi | Artículos para programadores, Java, Programación