19 de Mayo de 2011

Java Reflection (parte 2)

En nuestra anterior entrada, Java Reflection (parte 1), comentamos cómo obtener el tipo, constructores e instancias de una clase cuando no conocíamos en tiempo de compilación los detalles específicos de la misma.

En este artículo, comentaremos cómo obtener y manipular los atributos y métodos de una clase.

Como ejemplo que usaremos durante nuestra explicación, definimos la siguiente clase simple:

```
package com.test.model;

public class User()
{
        private String alias = null;
        public String name;
        public String address;

        public User(String name)
        {
                this.name = name;
        }

        public setAlias(String alias)
        {
                this.alias = alias;
        }

        public getAlias()
        {
                if (alias == null)
                {
                        return name;
                }
                else
                {
                        return alias;
                }
        }
}
```

Y supongamos que tenemos la clase almacenada en la variable `userClass`:

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

Atributos

Para acceder a los atributos públicos de una clase tenemos dos posibilidades. Si conocemos el nombre del atributo usaremos la siguiente instrucción:

```
Field userField = userClass.getField("name");
```

Como en los casos anteriores se nos devuelve toda la información sobre el atributo mediante una instancia al objeto correspondiente, en este caso de tipo `Field`.

Si la clase no tuviera ningún atributo público con ese nombre, el método `getField()` lanzará una excepción `NoSuchFieldException`.

Si por cualquier motivo no conocemos los nombres de los atributos, tenemos una forma de obtener todos los atributos públicos de una clase de la siguiente forma:

```
Field[] userFields = userClass.getFields();
```

La instrucción `getFields()` nos devuelve un `array` con un elemento de tipo `Field` por cada uno de los atributos públicos de la clase (en nuestro ejemplo, devolvería un `array` de 2 elementos: "`name`" y "`address`").

Las dos instrucciones mencionadas hasta el momento sirven sólo para acceder a los atributos públicos. Si lo que queremos es acceder a cualquier atributo (incluidos los privados), necesitaremos usar los métodos `Class.getDeclaredField(String name)` (para acceder sabiendo el nombre del atributo) y `Class.getDeclaredFields()` (que nos devolverá un `array` con todos los atributos declarados en la clase: "`alias`", "`name`" y "`address`").

> Debemos tener en cuenta que estos métodos sólo nos permiten acceder a los atributos declarados expresamente en la clase en cuestión, nunca aquellos declarados en superclases de la misma.

Ahora que disponemos de una instancia `Field` para un atributo, podemos proceder a manipularlo. Primero veremos cómo obtener información sobre el mismo. Podemos obtener el nombre del atributo mediante:

```
String fieldName = userField.getName();
```

También podemos averiguar el tipo del atributo mediante:

```
Object fieldType = userField.getType();
```

Si lo que nos interesa es el valor contenido en el atributo, deberemos usar:

```
Object fieldValue = userField.get(userInstance);
```

Como vemos, el método `Field.get()` recibe un parámetro que es la instancia del objeto en cuestión del que queremos averiguar el valor de su atributo. Si estuviéramos intentando acceder a un método estático, debemos llamar al método con `null` como parámetro.

Si lo que queremos es cambiar el valor del atributo, escribiremos:

```
userField.set(userInstance, value);
```

Donde `value` es el valor que queremos asignarle (que, evidentemente, debe ser del tipo correspondiente a ese atributo) y `userInstance` es la instancia del objeto al que queremos asignarle el valor. Al igual que con `get()`, en caso de tratarse de un atributo estático, el valor de `userInstance` debe ser `null`.

Como parece lógico, los métodos anteriores (`get()` y `set()`) sólo nos permiten manipular los atributos públicos de un objeto (aquellos a los que tengamos acceso desde el contexto de nuestro código). Pero Java Reflection nos permite manipular incluso aquellos atributos privados a los que no tendríamos acceso de forma normal. Para ello sólo hay que usar el método `Field.setAccessible(true)`, que deshabilita los chequeos de acceso para ese campo en particular (para Java Reflections sólo). De esta forma, si en nuestro ejemplo queremos modificar el valor del atributo privado "`alias`", sólo tendremos que hacer:

```
Object userInstance = userClass.getConstructor(new Class[]
{String.class}).newInstance(new Object[] {"José González"});
Field aliasField = userClass.getDeclaredField("alias");
aliasField.setAccessible(true);
aliasField.set(userInstance, "Pepe");
```

Métodos

De forma totalmente análoga a como accedimos a los atributos de una clase, podemos acceder a todos los métodos públicos de una clase mediante:

```
Method[] userMethods = userClass.getMethods();
```

Como podemos adivinar del ejemplo, los métodos de una clase se almacenan en un objeto de tipo `Method`.

Para acceder a un método específico no necesitamos saber sólo su nombre, si no que necesitamos saber el tipo y orden de los parámetros necesarios para su invocación (puesto que recordemos que la sobrecarga de operadores nos permite definir varios métodos con el mismo nombre y distintos parámetros). Así, para obtener el método `setAlias()` de nuestra clase de ejemplo, escribiríamos:

```
Method userMethod = userClass.getMethod("setAlias", new Class[] {String.class});
```

De la misma forma que ocurría con los atributos, estas instrucciones sólo nos devuelven los métodos públicos de una clase. Para poder acceder a los métodos privados deberemos usar respectivamente `Class.getDeclaredMethod(String name)` y `Class.getDeclaredMethods()`. También disponemos de un método `Method.setAccessible(true)` para poder manipular métodos privados de una clase.

Una vez tenemos el método deseado en nuestro objeto de tipo `Method`, procedemos a manipularlo. Para obtener el nombre y tipo de retorno usaremos métodos análogos a los que usamos para el caso de los atributos:

```
String methodName = userMethod.getName();
Object methodType = userMethod.getReturnType();
```

Finalmente, podemos invocar el método deseado mediante la orden `Method.invoke()`:

```
Object userInstance = userClass.getConstructor(new Class[]
{String.class}).newInstance(new Object[] {"José González"});
Method setAliasMethod = userClass.getMethod("setAlias", String.class);
Method getAliasMethod = userClass.getMethod("getAlias", null);
setAliasMethod.invoke(userInstance, "Pepe");
String newAlias = getAliasMethod.invoke(userInstance, null);
```

El método `Method.invoke()` debe recibir como primer parámetro la instancia en particular de la que queremos invocar el método (`null` si es un método estático) y los parámetros del método que queremos invocar (`null` o `array` vacío si no dispone de parámetros).

Puedes leer más sobre este tema en nuestro artículo Java Reflection (parte 3).

Creado por Santi | Artículos para programadores, Java, Programación