Introducción al API Reflection (Reflexión) de Java.

Fecha de creación: **27.07.2004**
Revisión **1.0** (**27.07.2004**)

Eneko González Benito (Keko)
enekog AT euskalnet DOT net

Introducción

[1] El API Reflection es una herramienta muy poderosa que nos permite realizar en Java cosas que en otros lenguajes es imposible. Sin embargo, y a pesar de su potencial, es un API bastante desconocido, sobre todo para los principiantes en el mundo Java. A lo largo de este árticulo, vamos a intentar comprender las posibilidades que nos ofrece este API, introduciéndonos poco a poco en su uso.

[2] Todos sabemos que un buen diseño es **muy importante** (si no imprescindible) en cualquier programa. Para ello, estamos acostumbrados a utilizar herencia, interfaces, etc. en nuestros programas. Sin embargo, la infinita variedad de problemas a los que no podemos tener que enfrentar, hace que estas técnicas no siempre sean suficientes para nuestros propósitos.

[3] Pongamos, por ejemplo, que estamos trabajando en un programa que nos ayude a *debuggear* otros programas, describiendo el contenido de cualquier objeto que queramos. ¿Como hacemos que nuestro programa trabaje con el contenido de dichos objetos? Lo primero que se nos puede ocurrir es crear un interfaz que deben cumplir los objetos *describibles*, pero ... ¿que ocurre cuando esos objetos no han sido creados por nosotros? Aqui es donde entra en juego el API Reflection.

Trabajando con clases.

[1] Todos los objetos en java heredan de la clase *java.lang.Object* y por ello estan dotados de un método *getClass*, cuya firma es *public final Class getClass()*. Este método nos devuelve un objeto *java.lang.Class*, que va a ser nuestro punto de entrada al API Reflection. Si alguna vez has cargado clases dinamicante en Java, es posible que conozcas esta clase por su método *forName* que se utiliza para cargar e instanciar clases del classpath, pero a lo mejor no te has parado a mirar otros métodos muy interesantes que tiene, como los siguientes.

- *java.lang.Class forName(String className)*: Carga una clase **del classpath** a partir de su

nombre (nombre completo, con todos los paquetes. Si la clase no se puede cargar, porque no se encuentra en el classpath, se lanzará una *java.lang.ClassNotFoundException*.

- *java.lang.reflect.Field getField(String name)*: Devuelve un campo **público** de la clase, a partir de su nombre. Si la clase no contiene ningún campo con ese nombre, se comprueban sus superclases recursivamente, y en caso de no encontrar finalmente el campo, se lanzará la excepcion *java.lang.NoSuchFieldException*.
- *java.lang.reflect.Field[] getFields()*: Devuelve un array con todos los campos **públicos** de la clase, y de sus superclases.
- *java.lang.reflect.Method getMethod(String name, Class[] parameterTypes)*: Devuelve un método **público** de la clase, a partir de su nombre, y de un array con las clases de los parámetros del método. Si la clase no contiene ningún método con ese nombre y esos parámtetros, se lanzará la excepcion *java.lang.NoSuchMethodException*.
- *java.lang.reflect.Method[] getMethods()*: Devuelve un array con todos los métodos **públicos** de la clase, y de sus superclases.
- *java.lang.Class[] getInterfaces()*: Devuelve un array con todos los interfaces que implementa la clase.

[2] Vamos a ver un ejemplo muy sencillo de como utilizar de algunos de esos métodos. Primero, necesitamos una clase sobre la que vamos a trabajar, como puede ser la siguiente.

```
public class EjemploBean {

  public String nombre = "Keko";
  private String email = "keko@miservidor.es";

  private void setNombre(String s) {
    nombre = s;
  }

  protected String getNombre() {
    return nombre;
  }

  public void setEmail(String s) {
    email = s;
  }

  public String getEmail() {
    return email;
  }
}
```

También necesitamos la clase que va a realizar el trabajo, cuyo código podemos ver a continuación.

```
import java.lang.reflect.*;

public class EjemploReflection {
```

```
  public static void main(String arg[]) {

    Class clase;
    Field campo, campos[];
    Method metodo, metodos[];

    try {
      // Cargamos la clase
      clase = Class.forName("EjemploBean");

      // Recorremos los campos
      System.out.println("Lista de campos:\n");
      campos = clase.getFields();
      for (int i=0; i < campos.length; i++) {
        campo = campos[i];
        System.out.println("\t" + campo.getName());
      }

      // Recorremos los metodos
      System.out.println("\nLista de metodos:\n");
      metodos = clase.getMethods();
      for (int i=0; i < metodos.length; i++) {
        metodo = metodos[i];
        System.out.println("\t" + metodo.getName());
      }
    } catch (ClassNotFoundException e) {
    System.out.println("No se ha encontrado la clase. " + e);
    }
  }
}
```

[3] A continuación, si ejecutamos la clase *EjemploReflection*, el resultado que vamos a obtener es el siguiente.

```
Lista de campos

        nombre

Lista de metodos

        setEmail
        getEmail
        hashCode
        getClass
        wait
        wait
        wait
        equals
        notify
```

```
        notifyAll
        toString
```

¿Qué ha ocurrido? ¿Dónde está el campo *email*? ¿Y los métodos *getNombre* y *setNombre*? ¿De dónde han salido esos otros métodos (*hashCode*, *getClass*, etc.)? Como ya hemos dicho antes, los métodos de la API Reflection nos permiten acceder a los campos y métodos **públicos** (por eso no aparecen *email*, *getNombre* y *setNombre*) y los métodos *hashCode*, *getClass*, etc. son métodos de la clase *java.lang.Object*, de la que hereda nuestra clase. Sencillo, ¿o no? Pues vamos a intentar complicarlo un poco más, obteniendo más información de los campos y los métodos.

Trabajando con clases. Campos

[1] Si miramos en la API de Reflection, la clase *java.lang.reflect.Field* podemos ver muchos métodos que nos van a permitir acceder a los campos de una clase, acceder a su valor en un objeto determinado, e incluso modificar su valor. Vamos a ver algunos de ellos.

- *public String getName()*: Devuelve el nombre del campo.
- *public Class getType()*: Devuelve la clase del campo.
- *public Object get(Object obj)*: Devuelve el valor del campo en un objeto.
- *public void set(Object obj, Object value)*: Asigna un valor al campo en un objeto.

Como podemos ver, los métodos *getName* y *getType* se aplican sobre la clase, pero los métodos *get* y *set* se aplican sobre un determinado objeto. Estos métodos los veremos en el último apartado.

Trabajando con clases. Métodos

[1] Vamos a fijarnos ahora en la clase *java.lang.reflect.Method*, en la que podemos encontrar, entre otros, los siguientes métodos.

- *public String getName()*: Devuelve el nombre del método.
- *public Class[] getParameterTypes()*: Devuelve un array con las clases de los parámetros del método.
- *public Class[] getExceptionTypes()*: Devuelve un array con las clases de las excepciones que puede lanzar el método.
- *public Class getReturnType())*: Devuelve la clase del valor que devuelve el método.
- *public Object invoke(Object obj, Object[] args)*: Ejecuta el método sobre un objeto, pasándole los parámetros necesarios, y devuelve su resultado.

Como en el caso anterior, ciertos métodos son aplicables directamente sobre la clase, mientras que otros (el método *invoke*) se aplica sobre un objeto concreto.

[2] Vamos a ver con un ejemplo como utilizar dichos métodos.

```
import java.lang.reflect.*;
```

```
public class EjemploReflection2 {

  public static void main(String arg[]) {

    Class clase;
    Field campo, campos[];
    Method metodo, metodos[];
    try {
      // Cargamos la clase
      clase = Class.forName("EjemploBean");

      // Recorremos los campos
      System.out.println("Lista de campos:\n");
      campos = clase.getFields();
      for (int i=0; i < campos.length; i++) {
        campo = campos[i];
        System.out.println("\t" + campo.getName() + " (" + campo.getType().getName
() + ")");
      }

      // Recorremos los metodos
      System.out.println("\nLista de metodos:\n");
      metodos = clase.getMethods();
      for (int i=0; i < metodos.length; i++) {
        metodo = metodos[i];
        System.out.print("\t" + metodo.getName() + " (");

        // Recorremos los parametros del metodo
        Class parametros[] = metodo.getParameterTypes();
        for (int j=0; j < parametros.length; j++) {
          System.out.print(parametros[j].getName());
          if (j < parametros.length-1) System.out.print(", ");
        }

        System.out.print(") = " + metodo.getReturnType().getName());

        // Recorremos las excepciones del metodo
        Class excepciones[] = metodo.getExceptionTypes();
        System.out.print(" [");
        for (int j=0; j < excepciones.length; j++) {
          System.out.print(excepciones[j].getName());
          if (j < excepciones.length-1) System.out.print(", ");
        }
        System.out.println("]");
      }
    } catch (ClassNotFoundException e) {
      System.out.println("No se ha encontrado la clase. " + e);
    }
  }
```

```
}
```

Ejecutamos esta clase, y el resultado es el siguiente.

```
Lista de campos:

        nombre (java.lang.String)

Lista de metodos:

        setEmail (java.lang.String) = void []
        getEmail () = java.lang.String []
        hashCode () = int []
        getClass () = java.lang.Class []
        wait () = void [java.lang.InterruptedException]
        wait (long) = void [java.lang.InterruptedException]
        wait (long, int) = void [java.lang.InterruptedException]
        equals (java.lang.Object) = boolean []
        notify () = void []
        notifyAll () = void []
        toString () = java.lang.String []
```

Ya tenemos una descripción detallada de nuestra clase *EjemploBean*. Sin embargo, lo habitual es que queramos trabajar con objetos, y no con sus clases, utilizando los métodos que ya hemos visto. Vamos con ello.

Trabajando con objetos.

[1] Ya hemos visto parte de los métodos del API Reflection, que nos permitirían acceder a los campos y métodos de una clase, y algunos que también nos permiten acceder a campos y métodos de un objeto concreto. Vamos a verlo con varios ejemplos.

[2] Para simplificarlo un poco, vamos a ir por partes. Primero, vamos a acceder directamente a los campos **públicos**, en el siguiente ejemplo.

```
import java.lang.reflect.*;

public class EjemploReflection3 {

  public static void main(String arg[]) {

    Class clase;
    Object objeto;
    Field campo, campos[];
    String valor;

    try {
      // Cargamos la clase
```

```
      clase = Class.forName("EjemploBean");

      // Instanciamos un objeto de la clase
      try {
        objeto = clase.newInstance();

        // Recorremos los campos
        System.out.println("Lista de campos:\n");
        campos = clase.getFields();
        for (int i=0; i < campos.length; i++) {
          campo = campos[i];

          // Leemos su valor
          valor = (String) campo.get(objeto);
          System.out.println("\t" + campo.getName() + " = " + valor + " (" +
campo.getType().getName()+ ")");

          // Cambiamos su valor
          valor += " nuevo";
          campo.set(objeto, valor);
          System.out.println("\tNuevo valor: " + campo.getName() + " = " + valor);
        }
      } catch (InstantiationException e) {
        System.out.println("Error al instanciar el objeto. " + e);
      } catch (IllegalAccessException e) {
        System.out.println("Error al instanciar el objeto. " + e);
      }
    } catch (ClassNotFoundException e) {
      System.out.println("No se ha encontrado la clase. " + e);
    }
  }
}
```

Este es el resultado de ejecutar la clase anterior.

```
Lista de campos:

        nombre = Keko (java.lang.String)
        Nuevo valor: nombre = Keko nuevo
```

Como se puede ver, no solo hemos podido acceder a los campos de un objeto, sino que hemos podido modificar su valor de una forma bastante sencilla.

[3] Con lo visto hasta ahora podemos acceder al contenido de los campos públicos de un objeto. Sin embargo, todos sabemos que lo usual (y recomendable) es que los campos de un objeto (de su clase) sean privados, y que tengamos disponibles unos métodos accesores (*getters*) y modificadores (*setters*) que operan sobre esos campos. Vamos a ver con un ejemplo, como también podemos ejecutar dichos métodos.

```
import java.lang.reflect.*;

public class EjemploReflection4 {

  public static void main(String arg[]) {

    Class clase;
    Object objeto;
    Method metGetEmail, metSetEmail;
    String resultado;
    Class[] clasesParamSetEmail;
    Object[] paramSetEmail;

    try {
      // Cargamos la clase
      clase = Class.forName("EjemploBean");

      // Instanciamos un objeto de la clase
      try {
        objeto = clase.newInstance();

        try {
          // Accedemos al metodo getEmail, sin parametros
          metGetEmail = clase.getMethod("getEmail", null);
          resultado = (String) metGetEmail.invoke(objeto, null);
          System.out.println("getEmail() = " + resultado);

          // Accedemos al metodo setEmail, con un parametro (String)
          clasesParamSetEmail = new Class[1];
          clasesParamSetEmail[0] = Class.forName("java.lang.String");
          metSetEmail = clase.getMethod("setEmail", clasesParamSetEmail);
          paramSetEmail = new Object[1];
          paramSetEmail[0] = resultado + " nuevo";
          metSetEmail.invoke(objeto, paramSetEmail);
          System.out.println("setEmail()");

          // Volvemos a llamar al metodo getEmail, sin parametros
          resultado = (String) metGetEmail.invoke(objeto, null);
          System.out.println("getEmail() = " + resultado);

        } catch (NoSuchMethodException e) {
          System.out.println("Error al acceder al metodo. " + e);
        } catch (SecurityException e) {
          System.out.println("Error al acceder al metodo. " + e);
        } catch (InvocationTargetException e) {
          System.out.println("Error al ejecutar el metodo. " + e);
        }
      } catch (InstantiationException e) {
        System.out.println("Error al instanciar el objeto. " + e);
      } catch (IllegalAccessException e) {
```

```
        System.out.println("Error al instanciar el objeto. " + e);
      }
    } catch (ClassNotFoundException e) {
      System.out.println("No se ha encontrado la clase. " + e);
    }
  }
}
```

Y el resultado de ejecutarlo.

```
getEmail() = keko@miservidor.es
setEmail()
getEmail() = keko@miservidor.es nuevo
```

Conclusión

[1] Hemos visto, con varios ejemplos, parte de las posibilidades del API Reflection de Java. Este API es más extenso de lo que aquí hemos visto, pero el objetivo de este artículo es hacer un introducción al API, para que cada uno investigue con mayor profundidad cuando lo necesite. Como siempre, lo mejor para aprender es leer toda la documentación[**1**] posible y hacer pruebas con nuestro propio código. Espero que este artículo nos sirva para perder un poco el miedo a este potente API.

[2] Me gustaría resaltar que los ejemplos aqui expuestos estan basados en un proyecto real, en el que trabajé para poder facilitar los logs de objetos cuyo método *toString* no daban demasiada información. Sin embargo, por si alguno quiere utilizarlo con el mismo fin, le recomiendo que no trate de reinventar la rueda, y que utilice la clase BeanUtils[**2**] de Apache Commons, cuyo método *describe* hace este trabajo de una manera excelente, amén de muchas más cosas.

[3] Por último, no quiero finalizar este artículo sin hacer una última observación. Hace poco leí un dicho en un libro, que decía así: **Para un hombre con un martillo, todo lo que ve, son clavos**. Con esto quiero decir que, si bien este API es muy potente, no es la solución a todos los problemas, y que utilizarlo puede convertirse en **matar moscas a cañonazos**. El código creado con este API tiende a ser bastante mas complejo que su equivalente *tradicional*, además de que su rendimiento es peor. Antes de utilizarlo, asegúrate de que es la mejor solución de la que dispones.

Recursos

[1] Tutorial (en inglés, pero muy completo) de Sun sobre el API Reflection, http://java.sun.com/docs/books/tutorial/reflect/

[2] Apache Commons BeanUtils, http://jakarta.apache.org/commons/beanutils/

Acerca del autor

Eneko González Benito (Keko)
Eneko lleva varios años programando en Java, y actualmente trabaja con Java en una consultoría en Bilbao. El poco tiempo libre que le deja el trabajo, se dedica a practicar deportes (senderismo, futbol, frontenis entre otros) y a leer todo lo que pasa por sus manos.