(/)

Establecer valor de campo con reflexión

Última modificación: 9 de agosto de 2020

por baeldung (https://www.baeldung.com/author/baeldung/)
(https://www.baeldung.com/author/baeldung/)

Java (https://www.baeldung.com/category/java/) +

Reflexión (https://www.baeldung.com/tag/reflection/)

1. Información general

En nuestro artículo anterior (/java-reflection-read-private-field-value) , discutimos cómo podríamos leer los valores de *los* campos *privados* de una clase diferente en Java. Sin embargo, puede haber escenarios en los que necesitemos establecer los valores de los campos, como en algunas bibliotecas donde no tenemos acceso a los campos.

En este tutorial rápido, discutiremos cómo podemos establecer los valores de los campos de una clase diferente en Java usando la API de Reflection (/java-reflection) .

Tenga en cuenta que usaremos la misma clase *Person* para los ejemplos aquí que usamos en nuestro artículo anterior (/java-reflection-read-private-field-value) .

2. Configuración de campos primitivos

Podemos establecer los campos que son primitivos usando los métodos *Field # setXxx* .

2.1. Configuración de campos enteros

Podemos usar los *métodos setByte, setShort* , s *etInt* y *setLong* para establecer los campos *byte* , *short* , *int* y *long* , respectivamente:

```
@Test
public void whenSetIntegerFields_thenSuccess()
  throws Exception {
    Person person = new Person();

    Field ageField = person.getClass()
        .getDeclaredField("age");
    ageField.setAccessible(true);

    byte age = 26;
    ageField.setByte(person, age);
    Assertions.assertEquals(age, person.getAge());

    Field uidNumberField = person.getClass()
        .getDeclaredField("uidNumber");
    uidNumberField.setAccessible(true);

    short uidNumber = 5555;
    uidNumberField.setShort(person, uidNumber);
    Assertions.assertEquals(uidNumber, person.getUidNumber());

    Field pinCodeField = person.getClass()
        .getDeclaredField("pinCode");
    pinCodeField.setAccessible(true);

    int pinCode = 411057;
    pinCodeField.setInt(person, pinCode);
    Assertions.assertEquals(pinCode, person.getPinCode());

    Field contactNumberField = person.getClass()
        .getDeclaredField("contactNumber");
    contactNumberField.setAccessible(true);

    long contactNumber = 123456789L;
    contactNumberField.setLong(person, contactNumber);
    Assertions.assertEquals(contactNumber, person.getContactNumber());

}
```

También es posible realizar unboxing (/java-wrapper-classes#autoboxing-and-unboxing) con tipos primitivos:

```
@Test
public void whenDoUnboxing_thenSuccess()
  throws Exception {
    Person person = new Person();

    Field pinCodeField = person.getClass()
        .getDeclaredField("pinCode");
    pinCodeField.setAccessible(true);

    Integer pinCode = 411057;
    pinCodeField.setInt(person, pinCode);
    Assertions.assertEquals(pinCode, person.getPinCode());
}
```

Los métodos s *etXxx* para tipos de datos primitivos también admiten el estrechamiento (/java-primitive-conversions#widening-primitive-conversions) :

```
@Test
public void whenDoNarrowing_thenSuccess()
  throws Exception {
    Person person = new Person();

    Field pinCodeField = person.getClass()
        .getDeclaredField("pinCode");
    pinCodeField.setAccessible(true);

    short pinCode = 4110;
    pinCodeField.setInt(person, pinCode);
    Assertions.assertEquals(pinCode, person.getPinCode());
}
```

2.2. Configuración de campos de tipo flotante

Para configurar los campos *flotante* y *doble* , necesitamos usar los métodos *setFloat* y *setDouble* , respectivamente:

```
@Test
public void whenSetFloatingTypeFields_thenSuccess()
  throws Exception {
    Person person = new Person();

    Field heightField = person.getClass()
        .getDeclaredField("height");
    heightField.setAccessible(true);

    float height = 6.1242f;
    heightField.setFloat(person, height);
    Assertions.assertEquals(height, person.getHeight());

    Field weightField = person.getClass()
        .getDeclaredField("weight");
    weightField.setAccessible(true);

    double weight = 75.2564;
    weightField.setDouble(person, weight);
    Assertions.assertEquals(weight, person.getWeight());
}
```

2.3. Configuración de campos de caracteres

Para establecer las *carbonilla* campos, podemos utilizar el *setChar* método:

```
@Test
public void whenSetCharacterFields_thenSuccess()
  throws Exception {
    Person person = new Person();

    Field genderField = person.getClass()
        .getDeclaredField("gender");
    genderField.setAccessible(true);

    char gender = 'M';
    genderField.setChar(person, gender);
    Assertions.assertEquals(gender, person.getGender());
}
```

2.4. Configuración de campos booleanos

De manera similar, podemos usar el método *setBoolean* para establecer el campo *booleano* :

```
@Test
public void whenSetBooleanFields_thenSuccess()
  throws Exception {
    Person person = new Person();

    Field activeField = person.getClass()
        .getDeclaredField("active");
    activeField.setAccessible(true);

    activeField.setBoolean(person, true);
    Assertions.assertTrue(person.isActive());
}
```

3. Establecer campos que sean objetos

Podemos establecer los campos que son objetos usando el método *Field # set* :

```
@Test
public void whenSetObjectFields_thenSuccess()
  throws Exception {
    Person person = new Person();

    Field nameField = person.getClass()
        .getDeclaredField("name");
    nameField.setAccessible(true);

    String name = "Umang Budhwar";
    nameField.set(person, name);
    Assertions.assertEquals(name, person.getName());
}
```

4. Excepciones

Ahora, analicemos las excepciones que la JVM puede generar al configurar los campos.

4.1. *Argumento de excepción ilegal*

La JVM arrojará *IllegalArgumentException* si usamos un mutador *setXxx* que es incompatible con el tipo del campo de destino . En nuestro ejemplo, si escribimos *nameField.setInt (person, 26)* , la JVM lanza esta excepción ya que el campo es del tipo String .

```
@Test
public void givenInt_whenSetStringField_thenIllegalArgumentException()
  throws Exception {
    Person person = new Person();
    Field nameField = person.getClass()
        .getDeclaredField("name");
    nameField.setAccessible(true);

    Assertions.assertThrows(IllegalArgumentException.class, () -> nameField.setInt(person, 26));
}
```

Como ya hemos visto, los métodos s *etXxx* admiten el estrechamiento de los tipos primitivos. Es importante tener en cuenta que debemos proporcionar el objetivo correcto para que la reducción sea exitosa . De lo contrario, la JVM arroja una *IllegalArgumentException* :

```
@Test
public void givenInt_whenSetLongField_thenIllegalArgumentException()
  throws Exception {
    Person person = new Person();

    Field pinCodeField = person.getClass()
        .getDeclaredField("pinCode");
    pinCodeField.setAccessible(true);

    long pinCode = 411057L;

    Assertions.assertThrows(IllegalArgumentException.class, () -> pinCodeField.setLong(person, pinCode));
}
```

4.2. *IllegalAccessException*

Si intentamos establecer un campo *privado* que no tiene derechos de acceso , la JVM lanzará una *IllegalAccessException* . En el ejemplo anterior, si no escribimos la declaración *nameField.setAccessible (true)* , entonces la JVM lanza la excepción:

```
@Test
public void whenFieldNotSetAccessible_thenIllegalAccessException()
  throws Exception {
    Person person = new Person();
    Field nameField = person.getClass()
        .getDeclaredField("name");

    Assertions.assertThrows(IllegalAccessException.class, () -> nameField.set(person, "Umang Budhwar"));
}
```

5. Conclusión

En este tutorial, hemos visto cómo podemos modificar o establecer los valores de los campos privados de una clase de otra clase en Java. También hemos visto las excepciones que puede generar la JVM y sus causas.

Como siempre, el código completo para este ejemplo está disponible en GitHub (https://github.com/eugenp/tutorials/tree/master/core-java-modules/core-java-reflection-2) .

0 COMENTARIOS

CATEGORÍAS

PRIMAVERA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
DESCANSO (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SEGURIDAD (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCIA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
LADO DEL CLIENTE HTTP (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIE

TUTORIAL DE JAVA "VOLVER A LO BÁSICO" (/JAVA-TUTORIAL)
TUTORIAL DE JACKSON JSON (/JACKSON)
TUTORIAL DE HTTPCLIENT 4 (/HTTPCLIENT-GUIDE)
DESCANSO CON SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
TUTORIAL DE PERSISTENCIA DE PRIMAVERA (/PERSISTENCE-WITH-SPRING-SERIES)
SEGURIDAD CON SPRING (/SECURITY-SPRING)

ACERCA DE

SOBRE BAELDUNG (/ABOUT)
LOS CURSOS (HTTPS://COURSES.BAELDUNG.COM)
TRABAJOS (/TAG/ACTIVE-JOB/)
EL ARCHIVO COMPLETO (/FULL_ARCHIVE)
ESCRIBE PARA BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORES (/EDITORS)
NUESTROS COMPAÑEROS (/PARTNERS)
ANUNCIE EN BAELDUNG (/ADVERTISE)

TÉRMINOS DE SERVICIO (/TERMS-OF-SERVICE)
POLÍTICA DE PRIVACIDAD (/PRIVACY-POLICY)
INFORMACIÓN DE LA COMPAÑÍA (/BAELDUNG-COMPANY-INFO)
CONTACTO (/CONTACT)