Adnotacje w języku JAVA

Tomasz Kubik

Adnotacje w kodzie

- Rodzaj komentarza lub metadanych, które można zamieścić w kodzie
- Używane początkowo do generowania dokumentacji kodu przez javadoc
- Obecnie używane przez frameworki (np. Spring i Hibernate)
- Od Java 5 wprowadzono nowe możliwości
  - definiowanie własnych adnotacji
    - raz zdefiniowane mogą być używane wielokrotnie
    - rozpoczynają się znakiem @
  - programowe przetwarzanie adnotacji
    - w czasie kompilacji (narzędzia do prekompilacji)
    - w czasie działania programu (za pomocą Java Reflection API)

Uwagi o adnotacjach

- Siedem wbudowanych
  - w pakiecie `java.lang.annotation`:
    `@Retention`, `@Documented`, `@Target`, `@Inherited`
  - w pakiecie `java.lang`:
    `@Deprecated`, `@Override`, `@SuppressWarnings`
- Deklarowane są trzy rodzaje adnotacji
  - Marker Annotations, np.
    ```
    @Override
    ```
  - Single value Annotations, np.
    ```
    @MyAnnotation("par")
    ```
  - Full Annotations, np.
    ```
    @MyAnnotation(no=10,value="par")
    ```

Uwagi o adnotacjach

- Deklaracja podobna do deklaracji interfejsu z metodami

```
[Access Specifier] @interface <AnnotationName>
{
   DataType <MethodName>() [default value];
}
```

- „Interfejs” może mieć modyfikatory dostępu
- Metody w „interfejsie” powinny być bezargumentowe, bez klauzuli `throws`
- Wartościami zwracanymi przez metody mogą być jedynie:
  - typy podstawowe, `String`, `Class`, typy wyliczeniowe `enum`, adnotacje,
  - jednowymiarowe tablice powyższych typów
- Metody (jak i „interfejs”) mogą mieć modyfikatory dostępu `public abstract`
- Zmienne w adnotacjach mogą mieć przypisane wartości domyślne

Definicja własnej adnotacji

```
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.ElementType;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface MyAnnotation {
    public String name();
    public String value();
}
```

```
@Target({ElementType.TYPE, ElementType.FIELD})
  @Retention(RetentionPolicy.RUNTIME)
@interface XMLElement { String value() default ""; }

@Target(ElementType.FIELD)
  @Retention(RetentionPolicy.RUNTIME)
@interface XMLAttibute { String value() default ""; }

@interface ArrayExampleAnnotation { String[] value()
  default {}; }
```

Uwagi o adnotacjach

- Adnotacje pobierają z nazwą i wartością elementu

```
@SuppressWarnings(value="unchecked")
```

- Gdy w adnotacji jest jedynie `value`, to można zapisać skrótowo:

```
@SuppressWarnings("unchecked")
```

- Przekazanie wartości w tablicy polega na użyciu nawiasów klamrowych

```
@SuppressWarnings({"unchecked", "unused"})
```

Dyrektywa Retention

- `@Retention(RetentionPolicy.RUNTIME)`
  - korzysta z :

```
public enum RetentionPolicy
  extends Enum<RetentionPolicy>
```

ze stałymi:

- CLASS - adnotacje są zapisywane w pliku klasy przez kompilator, ale nie muszą być przechowywane przez VM w czasie pracy
- RUNTIME - adnotacje są zapisywane w pliku klasy przez kompilator i przechowywane przez VM w czasie, więc mogą one być odczytywane metodami refleksji.
- SOURCE - adnotacje są odrzucane przez kompilator.

Dyrektywa Target (1)

- `@Target(ElementType.TYPE)`
  - korzysta z :

    ```
    public enum ElementType
    extends Enum<ElementType>
    ```

    ze stałymi:
    - ANNOTATION_TYPE – deklaracji typu opisów
    - CONSTRUCTOR – deklaracja konstruktora
    - FIELD – deklaracja pola (włączając stałe enum)
    - LOCAL_VARIABLE – deklaracja zmiennej lokalnej
    - METHOD – deklaracja metody
    - MODULE – deklaracja modułu
    - PACKAGE – deklaracja pakietu
    - PARAMETER – deklaracja parametru
    - TYPE – deklaracja klasy, interfejsu (w tym adnotacji), lub enum
    - TYPE_PARAMETER – deklaracja typu parametru (generyczne)
    - TYPE_USE – deklaracja typu (jak TYPE) oraz typy parametru (jak TYPE_PARAMETER)

Dyrektywa Target (2)

- Brak dyrektywy Target powoduje, że adnotacja może być stosowana do wszystkiego (klas, metod, pól)
- Dyrektywa Target może być definiowana z listą typów

```
@Target({ElementType.TYPE,
  ElementType.FIELD})
```

Adnotacje wbudowane

```
java.lang.Overrides
@Target(ElementType.METHOD)
public @interface Overrides
{}

class A {
@Overrides
    public String toString(int i)
    {
        return "";
    } }
```

- method does not override a method from its superclass @Overrides

```
java.lang.annotation.Documented
@Documented
@Target(ElementType.ANNOTATION_TYPE)
public @interface Documented
{
}

java.lang.annotation.Deprecated
@Documented
@Retention(RetentionPolicy.SOURCE)
public @interface Deprecated
{
}
```

```
java.lang.annotation.Inherited
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
public @interface Inherited
{
}

java.lang.annotation.Retention
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
public @interface Retention
{
    RetentionPolicy value();
}
```

```
java.lang.annotation.Target
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
public @interface Target
{
    ElementType[] value();
}
```

Wykorzystanie adnotacji klasy w czasie działania (1)

```
import java.lang.annotation.Annotation;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.ElementType;

@MyAnnotation(name="someName",  value = "someValue")
public class MyClass {
  MyClass(){
     Class aClass = MyClass.class;
     //Dostęp do adnotacji w czasie działania
     Annotation[] annotations = aClass.getAnnotations();
     for(Annotation annotation : annotations){
       if(annotation instanceof MyAnnotation){
    MyAnnotation myAnnotation = (MyAnnotation) annotation;
    System.out.println("name: " + myAnnotation.name());
    System.out.println("value: " + myAnnotation.value());
       } } } }
```

Wykorzystanie adnotacji klasy w czasie działania (2)

```
Class aClass = MyClass.class;
//Dostęp do specyficznej adnotacji w czasie działania
Annotation annotation =
   aClass.getAnnotation(MyAnnotation.class);
if(annotation instanceof MyAnnotation){
    MyAnnotation myAnnotation = (MyAnnotation) annotation;
    System.out.println("name: " + myAnnotation.name());
    System.out.println("value: " + myAnnotation.value());
}
```

Wykorzystanie adnotacji metody w czasie działania

```
import java.lang.reflect.Method;

class MyClass {
  // Adnotacja metody
  @MyAnnotation(name = "someName", value = "someValue")
  public static void myMethod(String str, int i) {  }

  public static void main(String[] a) {
    try {
      MyClass ob = new MyClass(); Class c = ob.getClass();
      Method m = c.getMethod("myMethod", String.class, int.class);
      // Dostęp do adnotacji metody w czasie działania
      Annotation[] annotations = m.getDeclaredAnnotations();
      for(Annotation annotation : annotations){
          if(annotation instanceof MyAnnotation){
              MyAnnotation myAnnotation = (MyAnnotation) annotation;
              System.out.println("name: " + myAnnotation.name());
              System.out.println("value: " + myAnnotation.value());
          }
    } } catch (NoSuchMethodException exc) {
      System.out.println("Method Not Found.");
    }
  }
}
```

Wykorzystanie adnotacji klasy w czasie działania (2)

```
Method m = c.getMethod("myMethod", String.class, int.class);
// Dostęp do specyficznej adnotacji metody w czasie działania
Annotation annotation = m.getAnnotation(MyAnnotation.class);

if(annotation instanceof MyAnnotation){
          MyAnnotation myAnnotation = (MyAnnotation) annotation;
          System.out.println("name: " + myAnnotation.name());
          System.out.println("value: " + myAnnotation.value());
       }
```

Wykorzystanie adnotacji parametrów metody w czasie działania (1)

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import java.lang.annotation.ElementType;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.PARAMETER)
public @interface MyAnnotation {
    public String name();
    public String value();
}
```

Wykorzystanie adnotacji parametrów metody w czasie działania (2)

```
import java.lang.annotation.Annotation;
import java.lang.reflect.Method;

public class MyClass{
  public static void myMethod(@MyAnnotation(name="name",value="value") String str,
  int i) {}
  public static void main(String[] a) {
    try {
      MyClass ob = new MyClass();
      Class c = ob.getClass();
      Method m = c.getMethod("myMethod", String.class, int.class);
      Annotation[][] parameterAnnotations = m.getParameterAnnotations();
      Class[] parameterTypes = m.getParameterTypes();
      int i=0;
      for(Annotation[] annotations : parameterAnnotations){
        Class parameterType = parameterTypes[i++];
        for(Annotation annotation : annotations){
          if(annotation instanceof MyAnnotation){
              MyAnnotation myAnnotation = (MyAnnotation) annotation;
              System.out.println("param: " + parameterType.getName());
              System.out.println("name : " + myAnnotation.name());
              System.out.println("value: " + myAnnotation.value());
          } } }
    } catch (NoSuchMethodException exc) {
      System.out.println("Method Not Found.");
    } }  }
```

```
param: java.lang.String
name : name
value: value
```

Wykorzystanie adnotacji pól w czasie działania (1)

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import java.lang.annotation.ElementType;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface MyAnnotation {
    public String name();
    public String value();
}
```

Wykorzystanie adnotacji pól w czasie działania (2)

```
import java.lang.annotation.Annotation;
import java.lang.reflect.Field;

public class MyClass{
  @MyAnnotation(name="name",  value = "value")
  public String myField = null;

  public static void main(String[] a) {
    try {
      MyClass ob = new MyClass();
      Class c = ob.getClass();
      Field f = c.getField("myField");
      Annotation[] annotations = f.getDeclaredAnnotations();
      for(Annotation annotation : annotations){
          if(annotation instanceof MyAnnotation){
              MyAnnotation myAnnotation = (MyAnnotation) annotation;
              System.out.println("name: " + myAnnotation.name());
              System.out.println("value: " + myAnnotation.value());
          } }
    } catch (NoSuchFieldException exc) {
      System.out.println("Field Not Found.");
    } } }
```

```
Annotation[] annotations = f.getAnnotations(MyAnnotation.class);
```

```
@TypeAnnotation class Foo {
@FieldAnnotation("here") public Object object;
@FieldAnnotation("there") public Object other;
@MethodAnnotation public Object
  setObject(@ParameterAnnotation final Object object){
this.object = object;    }
}
```

```
void setPropertyByAnnotationName(final Object object,
  final String name, final Object value) {
    final Field[] fields =
  object.getClass().getDeclaredFields();
    for (final Field field : fields) {
        final FieldAnnotation fieldAnnotation =
  field.getAnnotation(FieldAnnotation.class);
        if(fieldAnnotation!=null &&
  name.equals(fieldAnnotation.value()){
            field.set(object, value);
        }
    }
}

Foo foo = new Foo();
setPropertyByAnnotationName(foo, "here", "this is
  set!");
setPropertyByAnnotationName(foo, "there",
  Calendar.getInstance());
```

```
void callMethodByAnnotation(final Object object, final
  Object value) {
    final Method[] methods =
  object.getClass().getDeclaredMethods();
    for (final Method method : methods) {
        final MethodAnnotation methodAnnotation =
  method.getAnnotation(MethodAnnotation.class);
        if(fieldAnnotation!=null){
            method.invoke(object, value);
        }
    }
}

Foo foo = new Foo();
callMethodByAnnotation(foo, "this is set!");
```

```
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
public @interface ParameterAnnotation { String value();
   }

@TypeAnnotation class Foo {
    @FieldAnnotation("here") public Object object;

    @FieldAnnotation("there") public Object other;

    @MethodAnnotation public Object
  setObject(@ParameterAnnotation("inside") final Object
  object){
        this.object = object;
    }
}
```

```
public void callMethodByAnnotation(final Object object, final Map map) {
    final Method[] methods = object.getClass().getDeclaredMethods();
    for (final Method method : methods) {
        final MethodAnnotation methodAnnotation =
   method.getAnnotation(MethodAnnotation.class);
        if (methodAnnotation != null) {

            final Annotation[][] parameterAnnotations =
   method.getParameterAnnotations();
            final Object[] parameters = new
   Object[parameterAnnotations.length];
            for (int i = 0; i < parameterAnnotations.length; i++) {
                parameters[i] = null;

                final Annotation[] annotations = parameterAnnotations[i];
                for (final Annotation annotation : annotations) {
                    if (annotation instanceof ParameterAnnotation) {
                        parameters[i] = map.get(((ParameterAnnotation)
   annotation).value());
                    }
                }
            }
            method.invoke(object, parameters);
        }
    }
}
```

```
Map map = new Map();
map.put("inside", "this is set!");
Foo foo = new Foo();
callMethodByAnnotation(foo,map);
```