Стандартная библиотека Java: Reflection API

Алексей Владыкин

31 октября 2012

1. Аннотации
2. Reflection API

- Аннотации — это метаданные, сопровождающие исполняемый код
- В отличие от Javadoc, являются машиночитаемыми и могут быть доступны во время исполнения
- Примеры аннотаций:
  - `@Override`
  - `@Deprecated`
  - `@SuppressWarnings`

Что можно аннотировать

- Пакет
- Тип (класс, интерфейс, enum)
- Поле класса
- Метод, конструктор
- Параметр метода
- Локальная переменная

Где можно анализировать аннотации

- Во время компиляции
  (Annotation Processing API)
- В скомпилированных class-файлах
  (статический анализ FindBugs)
- Во время исполнения программы
  (Reflection API)

Создание аннотации

```
package ru.compscicenter.java2012.annotations;

import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Version {
    String value();
    String date() default "";
}
```

- Аннотация является особым видом класса:
  `extends Object implements Annotation`

Использование аннотации

```
package ru.compscicenter.java2012.annotations;

@Version(value = "1.3.44", date = "01.01.2011")
public class Component {
    // ...
}
```

- Экземпляр аннотации является объектом,
  у которого можно вызвать методы `value()` и `date()`
- Нельзя создать экземпляр аннотации вызовом `new`

Где применяются аннотации

- JUnit
- Java API for XML Binding (JAXB)
- Java Persistence API (JPA)

- Reflection API — программный интерфейс для получения информации об объектах и их классах во время исполнения программы
- Центральный класс — `java.lang.reflect.Class`
- Для каждого класса, загруженного в JVM, можно получить описывающий его экземпляр класса `Class`

Возможности Reflection API

- Получение списка конструкторов, методов и полей класса
- Создание экземпляров класса
- Вызов методов и чтение запись полей, в том числе закрытых

Откуда берутся Class'ы

- Получение класса по объекту:
  ```
  Class c1 = ref.getClass();
  ```
- Получение класса по имени:
  ```
  Class c2 = Class.forName("java.lang.Integer");
  ```
- Литералы:
  ```
  Class c3 = String.class;
  ```

Откуда берутся Class'ы

- Получение класса по имени, при этом класс может не входить в classpath запущенной JVM:

```
URL jarFileURL = new URL("file://electro.jar");

ClassLoader classLoader = new URLClassLoader(
        new URL[] {jarFileURL});

Class c4 = classLoader.loadClass("ElectroSolver");
```

Имя класса

	int[]	Object[]	Foo.Bar
getName()	[I	Ljava.lang.Object;	Foo$Bar
getCanonicalName()	int[]	java.lang.Object[]	Foo.Bar
getSimpleName()	int[]	Object[]	Bar

Аннотации

```
Version version =
        clazz.getAnnotation(Version.class);

String versionNumber = version.value();
String versionDate = version.date();
```

Иерархия классов

- `boolean isPrimitive()`
- `boolean isInterface()`
- `boolean isAnnotation()`
- `Class getSuperclass()`
- `Class[] getInterfaces()`

Специфика массивов

```
if (clazz.isArray()) {
    System.out.println(
            "Array of " + c.getComponentType());
}
```

Специфика enum

```
if (clazz.isEnum()) {
    System.out.println("Enum of:");
    for (Object e : clazz.getEnumConstants()) {
        System.out.println(e);
    }
}
```

Конструкторы

- Открытые конструкторы:
  `Constructor getConstructor(Class... types)`
  `Constructor[] getConstructors()`

- Все конструкторы:
  `Constructor getDeclaredConstructor(Class... types)`
  `Constructor[] getDeclaredConstructors()`

Вызов конструктора

```
Constructor constructor =
        clazz.getConstructor(String.class);

Object instance =
        constructor.newInstance("Hello World!");
```

Методы

- Открытые методы, в том числе унаследованные:
  `Method getMethod(String name, Class... types)`
  `Method[] getMethods()`

- Все методы, но только из текущего класса:
  `Method getDeclaredMethod(String name, Class... types)`
  `Method[] getDeclaredMethods()`

Вызов метода

```
Method method =
        clazz.getMethod("doSomething", int.class);

Object result =
        method.invoke(instance, 42);
```

Поля

- Открытые поля, в том числе унаследованные:
  `Field getField(String name)`
  `Field[] getFields()`

- Все поля, но только из текущего класса:
  `Field getDeclaredField(String name)`
  `Field[] getDeclaredFields()`

Чтение/запись поля

```
Field field = clazz.getDeclaredField("x");
field.setAccessible(true);

Object value = field.get(instance);

field.set(instance, null);
```

Что сегодня узнали

- Что такое аннотации в Java и как их можно использовать
- Что такое Reflection API и какие возможности он предоставляет