String Relevant Reflection APIs

Xilong Zhuo

October 13, 2018

1 Reflection

In this section we introduce the reflection APIs in Java and list all the APIs related to string.
Reflection can enable programs to examine or modify the runtime behavior of applications running in the Java virtual machine. Reflection APIs can be divided into 4 categories which are represented below respectively as *obtain a class*,*create a object*,*access to field* and *access to method*. String relevant APIs will be mark with "*" at the end of its name. Static methods showed in table below will be written as in italic.

1.1 Obtain a class

java.lang.Class is the entry point for all reflection operations. Table 1 shows several ways to get a *Class*.

1.2 Create a object

Reflection provides tow ways to create a object. They behaves very much like the *new* keyword. The details is showed in Table 2. The constructor object can be obtained by invoking *java.lang.Class.getConstructor(Class<? >... parameterTypes)* or *java.lang.Class.getConstructors()*.

1.3 Access to field

The *java.lang.reflect.Field* class provides methods for accessing type information and setting and getting values of a field on a given object. Details in table 3.

1.4 Access to method

The *java.lang.reflect.Method* class provides APIs to access information about a method's modifiers, return type, parameters, annotations, and thrown exceptions. It also be used to invoke methods.

API	**Description**
java.lang.Object.getClass()	only works for reference types which all inherit from *Object*
the **.class** Syntax	Confused here, At first it says the code in this site is correct, but then indicate that statement would produce a compile-time error. Here
*java.lang.Class.forName(String className)**	fully-qualified name of a class is available.The syntax for names of array classes is described by Class.getName()
*java.lang.Class.forName(String name, boolean initialize, ClassLoader loader)**	-
the **TYPE** field	for primitive types and *void*

java.lang.Class.getSuperclass()
java.lang.Class.getClasses()
java.lang.Class.getDeclaredClasses()
java.lang.Class.getDelaringClass()
java.lang.reflect.Field.getDeclaringClass()
java.lang.reflect.Method.getDeclaringClass()
java.lang.reflect.Constructor.getDeclaringClass()
	these methods can only be accessed if a class has already been obtained

Table 1: several ways to get a class

API	**Description**
java.lang.Class.newInstance()	It could only call the empty-parameter constructor and fails when that constructor can't be access(non-existent or private). But this is the commonest usage.
java.lang.reflect.Constructor.newInstance(Object... initargs)*	

Table 2: APIs to create a object

API	**Description**

java.lang.Class.getField(String name)*
java.lang.Class.getFields()
java.lang.Class.getDeclaredFields()
java.lang.Class.getDeclaredField(String name)*
	APIs with name including ”Declared” can return not only public field.

java.lang.reflect.Field.getModifiers()	
java.lang.reflect.Field.getName()	
java.lang.reflect.Field.set(Object obj, Object value)* java.lang.reflect.Field.setBoolean(Object obj, boolean z) setLong, setByte,...	the first parameter represents a certain object which these operation perform on.
java.lang.reflect.Field.get(Object obj) java.lang.reflect.Field.get...	get a value from certain object at runtime

Table 3: Access to method

API	**Description**

java.lang.Class.getMethod(String name, Class<? >... parameterTypes)
java.lang.Class.getMethods()
java.lang.Class.getDeclaredMethod(String name, Class<? >... parameterTypes)
java.lang.Class.getDeclaredMethods()
	same as field

java.lang.reflect.Method.getModifiers()	
java.lang.reflect.Method.getName()	
java.lang.reflect.Method.getParameters()	
java.lang.reflect.Method.invoke(Object obj, Object... args)*	invoking a method at runtime.

Table 4: access to method