

Java Reflection: A Basic Introduction

Reflection - This mechanism research data about the program during its execution. Reflection allows us to investigate the information about the fields , methods and constructors of classes. Reflection in Java by using Java Reflection API. This API consists of classes `java.lang` package and `java.lang.reflect`.

Using Java Reflection API interface , you can do the following:

- Determine the object class .
- Get information about modifiers classes , fields, methods, constructors, and superclasses .
- To find out what constants and methods belong to the interface.
- Create an instance of a class whose name is not known until runtime.
- Get and set the value of the object.
- Call the object method.
- Create a new array , the size and type of components which are not known until runtime programs.

Getting the object class type

```
MyClass a = new MyClass();  
Class aclass = a.getClass();
```

If we have an instance of the Class object we can get all possible information about this class , and even perform operations on it .The above method getClass () are often useful when there is then an object instance , but do not know what class this instance . If we have a class that is known at compile time type, get a copy of the class is even easier .

```
Class aclass = MyClass.class;  
Class iclass = Integer.class;
```

Getting the class name

```
Class c = myObject.getClass();  
String s = c.getName();
```

An object of type `String`, returned by `getName()`, will contain the fully qualified name of the class, that is, if `myObject` is the object type `Integer`, the result is the type `java.lang.Integer`.

Research modifiers class

```
Class c = obj.getClass();
int mods = c.getModifiers();
if (Modifier.isPublic(mods)) {
    System.out.println("public");
}
if (Modifier.isAbstract(mods)) {
    System.out.println("abstract");
}
if (Modifier.isFinal(mods)) {
    System.out.println("final");
}
```

Finding the superclass

```
Class c = myObj.getClass();  
Class superclass = c.getSuperclass();
```

To get all parent superclasses , you need to recursively call the method `getSuperclass ()`.

Research information on the method , the method call .

```
Class c = obj.getClass();
Method[] methods = c.getMethods();
for (Method method : methods) {
    System.out.println("Name: " +
        method.getName());
    System.out.println("Returned type: " +
        method.getReturnType().getName());

    Class[] paramTypes =
        method.getParameterTypes();
    System.out.print("Params's types: ");
    for (Class paramType : paramTypes) {
        System.out.print(" " +
            paramType.getName());
    }
    System.out.println();
}
```

Research information on the method , the method call .

Methods `getMethod ()` and `getMethods ()` return only the public methods , in order to get all the methods of the class regardless of the type of access you need `vospolzovatsya` methods `getDeclaredMethod ()` and `getDeclaredMethods ()`, which work just as well as their analogues (`getMethod ()` and `getMethods ()`).

Java Reflection Api interface allows you to dynamically invoke a method , even if at the time of compilation of this method is the name of the unknown (Class method names can be obtained by `getMethods ()` or `getDeclaredMethods ()`).

Research information on the method , the method call .

```
Class c = obj.getClass();
Class[] paramTypes = new Class[] {
    String.class, int.class };
Method method =
    c.getMethod("getCalculateRating",
    paramTypes);
Object[] args = new Object[] { new
    String("First Calculate"), new Integer(10)
    };
Double d = (Double) method.invoke(obj, args);
```

Download and dynamically create an instance of the class

```
Class c = Class.forName("Test");  
Object obj = c.newInstance();  
Test test = (Test) obj;
```

With the use of `Class.forName()` and `newInstance()` Class object can dynamically load and instantiate the class when the class name is not known until runtime.

`newInstance()` method returns a generic duplicate objects of type `Object`, so the last line we give the returned object of the type we need.