<http://blog.csdn.net/u011637069/article/details/52172518>

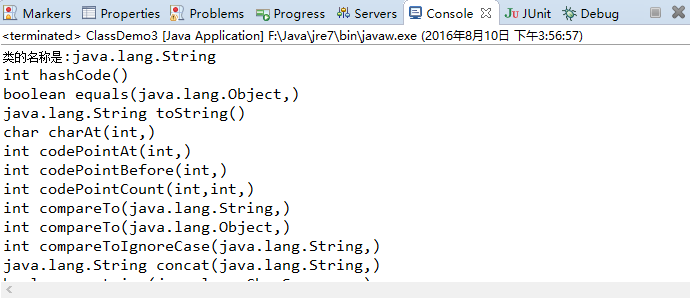
版权声明：本文为博主原创文章，未经博主允许不得转载。 <http://blog.csdn.net/u011637069/article/details/52172518>

Java反射之方法反射demo

**[java]** [view plain](http://blog.csdn.net/u011637069/article/details/52172518) [copy](http://blog.csdn.net/u011637069/article/details/52172518)

1. **package** reflect;
3. **import** java.lang.reflect.Method;
5. **public** **class** ClassDemo3 {
6. **public** **static** **void** main(String[] args) {
7. String s = "hello";
8. ClassUtil.printClassMethodMessage(s);
10. Integer n1 = 1;
11. ClassUtil.printClassMethodMessage(n1);
12. }
14. /\*\*
15. \* 获取方法
16. \* @param obj 该对象所属类的信息
17. \*/
18. **public** **static** **void** printClassMethodMessage(Object obj){
19. //要获取类的信息  首先要获取类的类类型
20. Class c = obj.getClass();//传递的是哪个子类的对象  c就是该子类的类类型
21. //获取类的名称
22. System.out.println("类的名称是:"+c.getName());
23. /\*
24. \* Method类，方法对象
25. \* 一个成员方法就是一个Method对象
26. \* getMethods()方法获取的是所有的public的函数，包括父类继承而来的
27. \* getDeclaredMethods()获取的是所有该类自己声明的方法，不问访问权限
28. \*/
29. Method[] ms = c.getMethods();//c.getDeclaredMethods()
30. **for**(**int** i = 0; i < ms.length;i++){
31. //得到方法的返回值类型的类类型
32. Class returnType = ms[i].getReturnType();
33. System.out.print(returnType.getName()+" ");
34. //得到方法的名称
35. System.out.print(ms[i].getName()+"(");
36. //获取参数类型--->得到的是参数列表的类型的类类型
37. Class[] paramTypes = ms[i].getParameterTypes();
38. **for** (Class class1 : paramTypes) {
39. System.out.print(class1.getName()+",");
40. }
41. System.out.println(")");
42. }
43. }
45. }

部分运行结果：



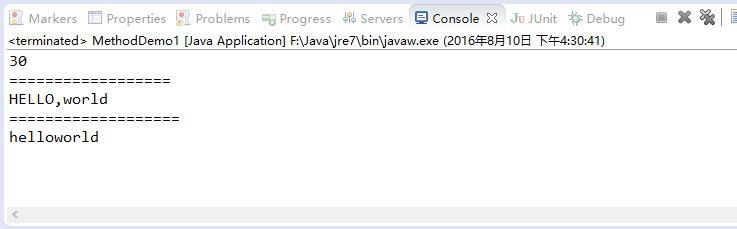
由代码可知，通过反射，可以根据类名（类全名）获取类中所有方法。

反射调用方法:

**[java]** [view plain](http://blog.csdn.net/u011637069/article/details/52172518) [copy](http://blog.csdn.net/u011637069/article/details/52172518)

1. **package** reflect2;
3. **import** java.lang.reflect.Method;
5. **public** **class** MethodDemo1 {
6. **public** **static** **void** main(String[] args) {
7. // 要获取print(int ,int )方法 1.要获取一个方法就是获取类的信息，获取类的信息首先要获取类的类类型
8. A a1 = **new** A();
9. Class c = a1.getClass();
10. /\*
11. \* 2.获取方法 名称和参数列表来决定 getMethod获取的是public的方法 getDelcaredMethod自己声明的方法
12. \*/
13. **try** {
14. // Method m = c.getMethod("print", new
15. // Class[]{int.class,int.class});
16. Method m = c.getMethod("print", **int**.**class**, **int**.**class**);
18. // 方法的反射操作
19. // a1.print(10, 20);方法的反射操作是用m对象来进行方法调用 和a1.print调用的效果完全相同
20. // 方法如果没有返回值返回null,有返回值返回具体的返回值
21. // Object o = m.invoke(a1,new Object[]{10,20});
22. Object o = m.invoke(a1, 10, 20);
23. System.out.println("==================");
24. // 获取方法print(String,String)
25. Method m1 = c.getMethod("print", String.**class**, String.**class**);
26. // 用方法进行反射操作
27. // a1.print("hello", "WORLD");
28. o = m1.invoke(a1, "hello", "WORLD");
29. System.out.println("===================");
30. // Method m2 = c.getMethod("print", new Class[]{});
31. Method m2 = c.getMethod("print");
32. // m2.invoke(a1, new Object[]{});
33. m2.invoke(a1);
34. } **catch** (Exception e) {
35. // TODO Auto-generated catch block
36. e.printStackTrace();
37. }
39. }
40. }
42. **class** A {
43. **public** **void** print() {
44. System.out.println("helloworld");
45. }
47. **public** **void** print(**int** a, **int** b) {
48. System.out.println(a + b);
49. }
51. **public** **void** print(String a, String b) {
52. System.out.println(a.toUpperCase() + "," + b.toLowerCase());
53. }
54. }

运行结果：



由运行结果可知，方法的调用不只是可以通过对象.来调用，还可以通过反射动态的调用。