# Java反射

在程序运行期间，Java运行时系统始终为所有的对象维护一个被称为运行时的类型标识。这个信息保存着每个对象所属的类足迹。虚拟机利用运行时信息选择相应的方法执行。可以通过专门的Java类访问这些信息。保存这些信息的类被称为Class。Object类中的getClass()方法将返回一个Class类型的实例。

如下方法可以获得一个Double的Class对象：

Class doubleC = Double.**class**;

Class doubleC1 = Class.*forName*("java.lang.Double");

Class doubleC2 = **new** Double(2.55).getClass();

注意，一个Class对象实际上表示的是一个类型，而这个类型未必一定是一种类。例如int和Double[]不是类，但int.class和Double[].class都是Class类型的对象。

从JDK1.5开始，Class类被参数化了。例如，Class<Integer>的类型是Integer.class。在大多数实际问题中，可以忽略类型参数，而使用原始的Class类。

虚拟机为每个类型管理一个Class对象。因此，可以利用==运算符来实现两个类对象的比较操作：

System.out.println(Integer.class == new Integer(4).getClass());

可以通过newInstance()方法来快速的创建一个类的实例：

Employee employee = Employee.class.newInstance();

newInstance()方法调用默认的构造器（无参数构造器）初始化新创建的对象。如果这个类没有默认的构造器，就会抛出异常。

在java.lang.reflect包中有三个类Field、Method和Constructor分别用于描述类的域、方法和构造器。这三个类都有一个叫做getName的方法，用来返回相应条目的名称。Field类有一个getType方法，用户返回描述域所属类型的Class类型对象。Method和Constructor类包含有能够报告参数类型的方法，Method类还可以报告返回类型。这三个类还有一个叫做getModifiers的方法，它将返回一个整型数字，用不同的位开关设置描述public和static这些修饰符的使用情况。还可以利用java.lang.reflect包中的Modifier类的静态方法根系getModifiers返回的整型数值。我们需要做的全部工作就是调用Modifier类的相应方法，并对getModifiers返回的整型数值进行分析，另外，还可以利用Modifier.toString方法将修饰符打印出来。

例子：打印一个类的全部信息

**public** **class** ReflectiveTest {

/\*\*

\* 打印类的全部信息

\* **@param** classStr

\* **@throws** ClassNotFoundException

\*/

**public** **static** **void** printClassInfo(String classStr) **throws** ClassNotFoundException {

Class c = Class.*forName*(classStr);

Class superC = c.getSuperclass();

System.*out*.println("class " + classStr);

**if**(superC != **null** && superC != Object.**class**) {

System.*out*.println(" extends " + superC.getName());

}

System.*out*.println(" {\n");

*printAllFields*(c);

System.*out*.println();

*printConstructors*(c);

System.*out*.println();

*printMethods*(c);

System.*out*.println("}");

}

/\*\*

\* 打印成员属性信息

\* **@param** c

\*/

**private** **static** **void** printAllFields(Class c) {

Field[] fields = c.getDeclaredFields(); // 返回其类和父类的所有域

**for**(Field f: fields) {

Class type = f.getType();

System.*out*.print(" " + Modifier.*toString*(f.getModifiers()));

System.*out*.println(" " + type.getName() + " " + f.getName() + ";");

}

}

/\*\*

\* 打印构造函数信息

\* **@param** c

\*/

**private** **static** **void** printConstructors(Class c) {

Constructor[] constructors = c.getDeclaredConstructors();

**for**(Constructor con : constructors) {

System.*out*.print(" " + Modifier.*toString*(con.getModifiers()));

System.*out*.print(" " + con.getName() + " (");

Class[] paramTypes = con.getParameterTypes();

**for**(**int** j = 0; j < paramTypes.length; j++) {

**if**(j > 0) {

System.*out*.print(", ");

}

System.*out*.print(paramTypes[j].getName());

}

System.*out*.println(");");

}

}

/\*\*

\* 打印成员方法信息

\* **@param** c

\*/

**private** **static** **void** printMethods(Class c) {

Method[] methods = c.getDeclaredMethods();

**for**(Method m : methods) {

Class retType = m.getReturnType();

// 返回值

System.*out*.print(" " + Modifier.*toString*(m.getModifiers()));

System.*out*.print(" " + retType.getName() + " " + m.getName() + "(");

// 参数类型

Class[] paramTypes = m.getParameterTypes();

**for**(**int** j = 0; j < paramTypes.length; j++) {

**if**(j > 0) {

System.*out*.print(", ");

}

System.*out*.print(paramTypes[j].getName());

}

System.*out*.println(");");

}

}

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException {

*printClassInfo*("java.lang.Double");

}

}

输出：

class java.lang.Double

extends java.lang.Number

{

public static final double POSITIVE\_INFINITY;

public static final double NEGATIVE\_INFINITY;

public static final double NaN;

public static final double MAX\_VALUE;

public static final double MIN\_NORMAL;

public static final double MIN\_VALUE;

public static final int MAX\_EXPONENT;

public static final int MIN\_EXPONENT;

public static final int SIZE;

public static final java.lang.Class TYPE;

private final double value;

private static final long serialVersionUID;

public java.lang.Double (java.lang.String);

public java.lang.Double (double);

public int hashCode();

public static native long doubleToRawLongBits(double);

public static long doubleToLongBits(double);

public static native double longBitsToDouble(long);

public volatile int compareTo(java.lang.Object);

public int compareTo(java.lang.Double);

public boolean equals(java.lang.Object);

public static java.lang.String toString(double);

public java.lang.String toString();

public static java.lang.String toHexString(double);

public static int compare(double, double);

public static java.lang.Double valueOf(double);

public static java.lang.Double valueOf(java.lang.String);

public static boolean isNaN(double);

public boolean isNaN();

public byte byteValue();

public double doubleValue();

public float floatValue();

public int intValue();

public boolean isInfinite();

public static boolean isInfinite(double);

public long longValue();

public short shortValue();

public static double parseDouble(java.lang.String);

}

**Class类中常用方法总结：**

static [Class](mk:@MSITStore:F:\编程相关\API\JDK_API_1_6_zh_CN%201.CHM::/java/lang/Class.html)<?> [**forName**](mk:@MSITStore:F:\编程相关\API\JDK_API_1_6_zh_CN%201.CHM::/java/lang/Class.html#forName(java.lang.String))([String](mk:@MSITStore:F:\编程相关\API\JDK_API_1_6_zh_CN%201.CHM::/java/lang/String.html) className)：返回与带有给定字符串名的类或接口相关联的 Class 对象。

String **getName**()：以 String 的形式返回此 Class 对象所表示的实体（类、接口、数组类、基本类型或 void）名称。

String **getSimpleName**()：返回源代码中给出的底层类的简称。

ClassLoader **getClassLoader**()： 返回该类的类加载器。

Constructor<?>[] **getDeclaredConstructors**()：返回此Class对象表示的类的所有构造方法。

Field[] **getDeclaredFields**()：返回此 Class 对象所表示的类或接口所有字段。

Method[] **getDeclaredMethods**()：返回此Class对象表示的类或接口声明的所有方法，但不包括继承的方法。

Constructor<?>[] **getConstructors**()：返回此Class对象表示的类的所有Public构造方法。

Field[] **getFields**()：返回此 Class 对象所表示的类或接口所有可访问（包括继承下来的）的Public字段。

Method[] **getMethods**()：返回此Class对象表示的类或接口声明的所有Public方法，且包括继承的方法。

Field[] **getDeclaredFields**()：返回此 Class 对象所表示的类或接口所有字段。

Method[] **getDeclaredMethods**()：返回此Class对象表示的类或接口声明的所有方法，但不包括继承的方法。

Constructor<?>[] **getConstructor**([Class](mk:@MSITStore:F:\编程相关\API\JDK_API_1_6_zh_CN%201.CHM::/java/lang/Class.html)<?>... parameterTypes)：根据参数类型返回此Class对象表示的类的对应的构造方法。

Field[] **getField**([String](mk:@MSITStore:F:\编程相关\API\JDK_API_1_6_zh_CN%201.CHM::/java/lang/String.html) name)：返回此 Class 对象所表示的类或接口所有可访问（包括继承下来的）的Public字段中对应该名称的字段。

Method[] **getMethod**([String](mk:@MSITStore:F:\编程相关\API\JDK_API_1_6_zh_CN%201.CHM::/java/lang/String.html) name, [Class](mk:@MSITStore:F:\编程相关\API\JDK_API_1_6_zh_CN%201.CHM::/java/lang/Class.html)<?>... parameterTypes)：返回此Class对象表示的类或接口声明的所有Public方法（包括继承的方法）中对应名称和参数的方法。

int **getModifiers**()：返回此类或接口以整数编码的 Java 语言修饰符。

Field[] getFields()方法将返回一个包含Field对象的数组，这些对象记录了该类或其超类的公有域。Field[] getDeclaredFields()方法也将返回包含Field对象的数组，这些对象记录了该类的全部域，如果类中没有域，或者Class对象描述的是基本数据类型或数组类型，那么这个方法将返回一个长度为0的数组。

Method[] getMethods()将返回所有的公有方法，包括从超类继承而来的公有方法。Method[] getDeclaredMethods()返回该类或接口的全部方法，但不包括由超类继承了的方法。

我们可以通过Class的public [T](mk:@MSITStore:E:\编程相关\API\JDK_API_1_6_zh_CN.CHM::/java/lang/Class.html) newInstance()或Constructor的public [T](mk:@MSITStore:E:\编程相关\API\JDK_API_1_6_zh_CN.CHM::/java/lang/reflect/Constructor.html) newInstance ([Object](mk:@MSITStore:E:\编程相关\API\JDK_API_1_6_zh_CN.CHM::/java/lang/Object.html)... initargs)方法来创建一个新的实例。

我们可以通过Field的public [Object](mk:@MSITStore:E:\编程相关\API\JDK_API_1_6_zh_CN.CHM::/java/lang/Object.html) get([Object](mk:@MSITStore:E:\编程相关\API\JDK_API_1_6_zh_CN.CHM::/java/lang/Object.html) obj)来获得指定对象上此 Field 表示的字段的值。如果该值是一个基本类型值，则自动将其包装在一个对象中。

我们可以通过Method的public [Object](mk:@MSITStore:E:\编程相关\API\JDK_API_1_6_zh_CN.CHM::/java/lang/Object.html) invoke([Object](mk:@MSITStore:E:\编程相关\API\JDK_API_1_6_zh_CN.CHM::/java/lang/Object.html) obj, [Object](mk:@MSITStore:E:\编程相关\API\JDK_API_1_6_zh_CN.CHM::/java/lang/Object.html)... args)来调用obj对象的该方法，如果底层方法是静态的，那么可以忽略指定的 obj 参数。该参数可以为 null。如果底层方法所需的形参数为 0，则所提供的 args 数组长度可以为 0 或 null。