使用反射也可以取得：

* 实现的全部接口
* public Class<?>[] getInterfaces()

确定此对象所表示的类或接口实现的接口。

@Test  
public void test4() {  
  
 Class clazz = Person.class;  
  
 Class[] interfaces = clazz.getInterfaces();  
 for (Class i : interfaces) {  
 System.***out***.println(i);  
 }  
}

* 所继承的父类
* public Class<? Super T> getSuperclass()

返回表示此Class所表示的实体（类、接口、基本类型）的父类Class。

@Test  
public void test1() {  
  
 Class clazz = Person.class;

Class superClass = clazz.getSuperclass();  
 System.***out***.println(superClass);  
}

* 全部的构造器
* public Constructor<T>[] getConstructors()

返回此Class对象所表示的类的所有public构造方法。

* public Constructor<T>[] getDeclaredConstructors()

返回此Class对象表示的类声明的所有构造方法。

* 全部的方法
* **public Method[] getDeclaredMethods()**

获取**运行时类本身**的**全部方法**。

Class clazz = Person.class;

Method[] methods1 = clazz.getDeclaredMethods();  
for (Method m: methods1) {  
 System.***out***.println(m);  
}

* **public Method[] getMethods()**

获取**运行时类及其父类**中所有的**public的方法**。

Class clazz = Person.class;  
  
Method[] methods = clazz.getMethods();  
for (Method m: methods) {  
 System.***out***.println(m);  
}

* Method类中：
* public Class<?> getReturnType()取得全部的返回值
* public Class<?>[] getParameterTypes()取得全部的参数
* public int getModifiers()取得修饰符
* public Class<?>[] getExceptionTypes()取得异常信息
* **全部的Field**
* **public Field[] getFields()**

获取**运行时类中及其父类中**生命为**public的Field**。

Class clazz = Person.class;

//获得声明为public的属性  
Field[] fields = clazz.getFields();  
for (int i = 0; i < fields.length; i++) {  
 System.***out***.println(fields[i]);  
}

* **public Field[] getDeclaredFields()**

获取**运行时类本身**声明的**全部Field**。

Class clazz = Person.class;

//获得全部属性  
Field[] fields1 = clazz.getDeclaredFields();  
for (Field f : fields1) {  
 System.***out***.println(f.getName());  
}

* Field方法中：
* public int getModifiers()以整数形式返回此Field的修饰符。
* public Class<?> getType()得到Field的属性类型。
* public String getName()返回Field的名称。

/\*\*  
 \* 获取属性的各个部分的内容（权限修饰符、变量类型、变量名）  
 \*/  
@Test  
public void test2() {  
 Class clazz = Person.class;  
  
 Field[] fields1 = clazz.getDeclaredFields();  
 for (Field f : fields1) {  
  
 //1、获取权限修饰符  
 int i = f.getModifiers();  
 String m = Modifier.toString(i);  
 System.***out***.println(m);  
  
 //2、获取属性的变量类型  
 Class t = f.getType();  
 System.***out***.println(t.getName());  
  
 //3、获取属性名  
 System.***out***.println(f.getName());  
 }  
}

* Annotation相关
* get Annotation(Class<T> annotationClass)
* getDeclaredAnnotations()

/\*\*  
 \* 获取注解  
 \* 只有被声明RUNTIME的才能获取到  
 \*/  
@Test  
public void test6() {  
 Class clazz = Person.class;  
  
 Annotation[] anns = clazz.getAnnotations();  
 for (Annotation a:anns) {  
 System.***out***.println(a);  
 }  
}

* 泛型相关
* 获取父类泛型类型：Type getGenericSuperclass()
* 泛型类型：ParameterizedType
* 获取实际的泛型类型参数数组：getActualTypeArguments()

/\*\*  
 \* 获取带泛型的父类  
 \*/  
@Test  
public void test2() {  
  
 Class clazz = Person.class;  
 Type type1 = clazz.getGenericSuperclass();  
 System.***out***.println(type1);  
}  
  
/\*\*  
 \* 获取父类的泛型（重要）  
 \*/  
@Test  
public void test3() {  
  
 Class clazz = Person.class;  
  
 Type type1 = clazz.getGenericSuperclass();  
 ParameterizedType param = (ParameterizedType) type1;  
 Type[] ars = param.getActualTypeArguments();  
 System.***out***.println(((Class) ars[0]).getName());  
}

* 类所在的包
* Package getPackage()

@Test  
public void test5() {  
  
 Class clazz = Person.class;  
  
 Package pack = clazz.getPackage();  
 System.***out***.println(pack);  
}