我們可以通過反射,獲取對應的運行時類中所有屬性、方法、構造器、父類、接口、父類的泛型、包、註解、異常等…

典型代碼:

public void test1*(){* Class clazz = Person.class;  
 *//获取属性结构  
 //getFields():获取当前运行时类及其父类中声明为public访问权限的属性* Field*[]* fields = clazz.getFields*()*;  
 for*(*Field f : fields*){* System.*out*.println*(*f*)*;  
 *}* System.*out*.println*()*;  
 *//getDeclaredFields():获取当前运行时类中声明的所有属性。（不包含父类中声明的属性）* Field*[]* declaredFields = clazz.getDeclaredFields*()*;  
 for*(*Field f : declaredFields*){* System.*out*.println*(*f*)*;  
 *}  
}*

public void test1*(){* Class clazz = Person.class;  
 *//getMethods():获取当前运行时类及其所有父类中声明为public权限的方法* Method*[]* methods = clazz.getMethods*()*;  
 for*(*Method m : methods*){* System.*out*.println*(*m*)*;  
 *}* System.*out*.println*()*;  
 *//getDeclaredMethods():获取当前运行时类中声明的所有方法。（不包含父类中声明的方法）* Method*[]* declaredMethods = clazz.getDeclaredMethods*()*;  
 for*(*Method m : declaredMethods*){* System.*out*.println*(*m*)*;  
 *}  
}*

*/\*  
 获取构造器结构  
 \*/* @Test  
 public void test1*(){* Class clazz = Person.class;  
 *//getConstructors():获取当前运行时类中声明为public的构造器* Constructor*[]* constructors = clazz.getConstructors*()*;  
 for*(*Constructor c : constructors*){* System.*out*.println*(*c*)*;  
 *}* System.*out*.println*()*;  
 *//getDeclaredConstructors():获取当前运行时类中声明的所有的构造器* Constructor*[]* declaredConstructors = clazz.getDeclaredConstructors*()*;  
 for*(*Constructor c : declaredConstructors*){* System.*out*.println*(*c*)*;  
 *}  
 }  
 /\*  
 获取运行时类的父类  
 \*/* @Test  
 public void test2*(){* Class clazz = Person.class;  
 Class superclass = clazz.getSuperclass*()*;  
 System.*out*.println*(*superclass*)*;  
 *}  
 /\*  
 获取运行时类的带泛型的父类  
 \*/* @Test  
 public void test3*(){* Class clazz = Person.class;  
 Type genericSuperclass = clazz.getGenericSuperclass*()*;  
 System.*out*.println*(*genericSuperclass*)*;  
 *}  
 /\*  
 获取运行时类的带泛型的父类的泛型  
 代码：逻辑性代码 vs 功能性代码  
 \*/* @Test  
 public void test4*(){* Class clazz = Person.class;  
 Type genericSuperclass = clazz.getGenericSuperclass*()*;  
 ParameterizedType paramType = *(*ParameterizedType*)* genericSuperclass;  
 *//获取泛型类型* Type*[]* actualTypeArguments = paramType.getActualTypeArguments*()*;  
*// System.out.println(actualTypeArguments[0].getTypeName());* System.*out*.println*(((*Class*)*actualTypeArguments*[*0*])*.getName*())*;  
 *}  
 /\*  
 获取运行时类实现的接口  
 \*/* @Test  
 public void test5*(){* Class clazz = Person.class;  
 Class*[]* interfaces = clazz.getInterfaces*()*;  
 for*(*Class c : interfaces*){* System.*out*.println*(*c*)*;  
 *}* System.*out*.println*()*;  
 *//获取运行时类的父类实现的接口* Class*[]* interfaces1 = clazz.getSuperclass*()*.getInterfaces*()*;  
 for*(*Class c : interfaces1*){* System.*out*.println*(*c*)*;  
 *}  
 }  
 /\*  
 获取运行时类所在的包  
 \*/* @Test  
 public void test6*(){* Class clazz = Person.class;  
 Package pack = clazz.getPackage*()*;  
 System.*out*.println*(*pack*)*;  
 *}  
 /\*  
 获取运行时类声明的注解  
 \*/* @Test  
 public void test7*(){* Class clazz = Person.class;  
 Annotation*[]* annotations = clazz.getAnnotations*()*;  
 for*(*Annotation annos : annotations*){* System.*out*.println*(*annos*)*;  
 *}  
 }*