Java除了给我们提供在编译期得到类的各种信息之外，还通过反射让我们可以在运行期间得到类的各种信息。通过反射获取类的信息，得到类的信息之后，就可以获取以下相关类容：Class对象、构造器、变量、方法、私有变量与私有方法、注解、泛型、数组。

1. Class对象：JDK中提供了Class对象来保存类的信息，所以，反射第一步就是得到Class对象。
2. 获取Class对象：
   * + 1. 第一种，编写代码时就知道Class的名字：

Class list = ArrayList.class；

* + - 1. 第二种,编写时不知道类的名字，但在运行的时候，可以得到一个类名字符串：

Class list = Class.ForName(“java.util.ArrayList”);

1. 得到类的名字：
   * + 1. 第一种，得到类的全限定名

String name = list.getName();

System.out.println(name);

输出：java.util.ArrayList

* + - 1. 第二种，得到这个类的名字，不带包名

String simpleName = list.getSimpleName();

System.out.println(simpleName);

输出：ArrayList

1. 得到类的包名、父类和实现的接口：
   * + 1. 得到包信息

Package aPackage = list.getPackage();

System.out.println(aPackage);

输出：package java.util, Java Platform API Specification, version 1.8

* + - 1. 得到父类

Class superClass = list.getSuperclass();

System.out.println(superClass.getSimpleName);

输出:AbstractList

* + - 1. 得到实现的接口

Class [] classes = list.getInterfaces();

* + - 1. 得到修饰符

int modifiers = list.getModifiers();

Modifier.isAbstract(modifiers) 抽象的 false

Modifier.isFinal(modifiers) 最终的 false

Modifier.isInterface(modifiers) 接口的 false

Modifier.isNative(modifiers) 本地的 false

Modifier.isPrivate(modifiers) 私有的 false

Modifier.isProtected(modifiers) 保护的 false

Modifier.isPublic(modifiers) 公共的 true

Modifier.isStatic(modifiers) 静态的 false

Modifier.isStrict(modifiers) 严格的 false

Modifier.isSynchronized(modifiers) 线程安全的 false

Modifier.isTransient(modifiers) 不序列化的 false

Modifier.isVolatile(modifiers) 原子的 false

1. 构造器：利用java反射可以得到一个类的构造器，并根据构造器，在运行时动态的创建一个对象。
2. 获取所有构造器

Constructor [] constructors = list.getConstructors();

1. 根据访问参数获取指定构造器

Constructor constructor = list.getConstructor(int.class);

1. 获取构造器的参数

Class [] parameterType = constructor.getParameterTypes();

1. 根据构造器初始化对象

Object object = construct.newInstance(50);

1. 变量：利用java反射可以在运行时得到一个类的变量信息，并在创建对象后设置它的变量信息。
   1. 获取所有public变量（继承 public）：

Field[] fields = list.getFields();

* 1. 获取指定名称的public变量：

Field field = list.getField(String name);

* 1. 获取所有变量（自身 所有）：

Field fields = list.getDeclaredFields();

* 1. 获取指定名称的变量：

Field field = list.getDeclaredField();

* 1. 给反射的变量赋值：

field.set(list, Object... value);

如果不是public变量，需要先设置强制操作：field.setAccessible(true)；

* 1. 此方法不仅仅对成员变量有用，对静态变量也可以。如果事静态变量，传入null，不用传对象，也是可以的。

1. 方法：利用java反射可以在运行时得到一个类的方法信息，并在创建对象后去调用方法。
   1. 获取所有public方法（继承 public）：

Method[] methods = list.getMethods();

* 1. 获取指定名称的public方法：

Method method = list.getMethod(String name, Class...type);

* 1. 获取所有方法（自身 所有）：

Method[] methods = list.getDeclaredMethods()；

* 1. 获取指定方法：

Method method = list.getDeclaredMethod(String name, Class...type);

* 1. 调用反射的方法：

Method.invoke(list, Object... value)

如果不是public变量，需要先设置强制操作：method.setAccessible(true)；

* 1. 也可以获得反射方法的名称、简称、参数等，可以参照获取类的信息。
  2. 同Field对象一样，对于静态方法，同样传入null，调用静态方法。

1. 注解：java给我们提供了在运行时获取类的注解信息，可以得到类注解，方法注解，参数注解，变量注解。
   1. 获取当前成员(类、方法、变量)的注解，不包括继承的：

Annotation[] annotations = list.getDeclaredAnnotations();

Annotation annotation = list.getDeclaredAnnotation(Class annotation);

Annotation[] annotions = field.getDeclaredAnnotations();

Annotation annotation = field.getDeclaredAnnotation(Class annotation);

Annotation[] annotions = method.getDeclaredAnnotations();

Annotation annotation = method.getDeclaredAnnotation(Class annotation);

* 1. 获取当前成员(类、方法、变量)的所有注解，包括继承的：

注解上有@inherited才能被子类获取到

Annotation[] annotations = list.getAnnotations();

Annotation annotation = list.getAnnotation(Class annotation);

Annotation[] annotations = field.getAnnotations();

Annotation annotation = field.getAnnotation(Class annotation);

Annotation[] annotations = method.getAnnotations();

Annotation annotation = method.getAnnotation(Class annotation);

* 1. 获取参数上的注解：参数注解是比较麻烦的一项，第一步，先取method对象，调用getParameterAnnotations,但是这个返回值是一个二维数组，因为method对象有很多参数，每个参数可能有很多注解。

Annotation[][] annotationInParam = method.getParameterAnnotations();

Class[] params = method.getParameterTypes();

int i = 0;

for (Annotation[] annotations : annotationInParam) {

Class para = params[i++];

for(Annotation annotation : annotations) {

…………

}

}

1. 泛型：java泛型是通过擦除来实现的，很难直接得到泛型具体化参数类型信息，但是我们可以通过一种间接的形式利用反射得到泛型信息
   1. 获取方法的参数泛型类：

**//泛型变量可能有多个，所以获取到的是参数化类型数组**

Type[] genericTypes = method.getGenericParameterTypes;

for (Type genericType : genericType) {

System.out.println(genericType.getTypeName);

if (genericType instanceof ParameterizedType) {

ParameterizedType parameteriedType = (ParameterizedType) genericType;

**//可能有多个，例如map，所以获取到的是参数化类型数组**

Type[] types = parameterized.getActualTypeArguments();

for（Type type : types）{

Class actual = (Class) actual;

System.out.println(actual);

}

}

}

* 1. 获取方法返回泛型类型：

Type genericType = method.getGenericReturnType();

if (genericType instanceof ParameterizedType) {

ParameterizedType parameterizedType = (ParameterizedType) genericType;

Type[] types = parameterizedType.getActualTypeArguments();

for (Type type : types) {

Class actual = (Class) type;

}

}

* 1. 获取变量的泛型类型：

Type genericType = field.getGenericType();

if (genericType instanceof ParameterizedType) {

Type [] types = ((ParameterizedType) genericType).getActualTypeArguments();

for (Type type : types) {

Class actual = (Class) type;

}

}

1. 数组：java反射可以对数组进行操作，包括创建数组，访问数组种的值，得到一个数组对象。
   1. 在反射种使用Array这个类，是reflect包下面的。

//创建一个int类型的数组，长度为3

int [] intArray = (int []) Array.newInstance(int.class, 3);

//通过反射的形式，给数组赋值

for (int i = 0; i < intArray.length; i++) {

Array.set(intArray, i, i\*3);

}

//通过反射的形式，得到数组中的值

for (int i = 0; i < intArray.length; i++) {

Array.get(intArray, i);

}

* 1. 利用Class.forName的形式获取，写法比较奇怪。

//获取一个int数组

Clas clz = Class.forName(“[I”);

//获取一个字符串数组

Class clz = Class.forName(“[Ljava.lang.String;”);

这个forName中的字符串，[表示数组，I表示int，F表示float，D表示double等等，L的右边是类名，类型的右边是；

* 1. 的到数组对象后，就可以调用它的独特方法：

//获取成员类型名

clz.getComponentType().getTypeName();