****反射****

 JAVA反射机制是在运行状态中，对于任意一个类（class文件），都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象方法的功能称为java语言的反射机制。反射就是把java类中的各种成分映射成一个个的Java对象。

  所谓的框架就是对外提供一些接口，也就是功能扩展的标准，由实现类按照这个接口标准去实现。框架内部如果需要操纵这些实现类的对象完成某些操作，那么只需要把这些实现类的全名（包名+类名）写在某个配置文件中，框架代码只需要读取这个配置文件，就可以获取这个实现类的字节码文件，然后利用反射技术创建这个实现类的对象并且调用相应的方法完成一些操作。

  所谓的框架就是对外提供一些接口，也就是功能扩展的标准，由实现类按照这个接口标准去实现。框架内部如果需要操纵这些实现类的对象完成某些操作，那么只需要把这些实现类的全名（包名+类名）写在某个配置文件中，框架代码只需要读取这个配置文件，就可以获取这个实现类的字节码文件，然后利用反射技术创建这个实现类的对象并且调用相应的方法完成一些操作：

方式一：Object类中的getClass()方法。想要用这种方式，必须要明确具体的类，并创建对象。p1.getClass();

### 方式二： 任何数据类型（包括基本数据类型）都有一个“静态”的class属性。.class来获取其对应的Class对象。但是还是要明确用到类中的静态成员并导入依赖包。Person.class;

方式三：只要通过给定的类的字符串名称就可以获取该类，更为扩展。Class.forName(“className”);找寻该文件类文件，并加载进内存，并产生Class对象。

Person p = new Person();new时候，先根据被new的类的名称找寻该类的字节码文件，并加载进内存，并创建该字节码文件对象Class，并接着创建该字节文件的对应的Person对象。



其中这个Class对象很特殊。我们先了解一下这个Class类



Class 类的实例表示正在运行的 Java 应用程序中的类和接口。也就是jvm中有N多的实例每个类都有该Class对象。（包括基本数据类型）

Class 没有公共构造方法。Class 对象是在加载类时由 Java 虚拟机以及通过调用类加载器中的defineClass 方法自动构造的。也就是这不需要我们自己去处理创建，JVM已经帮我们创建好了。

## 2、获取构造方法并调用

1. public Constructor[] getConstructors()：所有"公有的"构造方法
2. public Constructor[] getDeclaredConstructors()：获取所有的构造方法(包括私有、受保护、默认、公有)
3. public Constructor getConstructor(Class... parameterTypes):获取单个的"公有的"构造方法，没有参数就是null或者不写
4. public Constructor getDeclaredConstructor(Class... parameterTypes):获取"某个构造方法"可以是私有的，或受保护、默认、公有；

constructor.setAccessible(**true**);//暴力访问(忽略掉访问修饰符)

调用构造方法：

Constructor.newInstance(Object... initargs)它的返回值是T类型，所以newInstance是创建了一个构造方法的声明类的新实例对象。并为之调用。

## 3、获取成员变量并调用

1. 1).Field[] getFields():获取所有的"公有字段"
2. 2).Field[] getDeclaredFields():获取所有字段，包括：私有、受保护、默认、公有；
3. 1).public Field getField(String fieldName):获取某个"公有的"字段；
4. 2).public Field getDeclaredField(String fieldName):获取某个字段(可以是私有的)
5. 设置字段的值：
6. Field-->public void set(Object obj,Object value):

field.setAccessible(**true**);//暴力反射，解除私有限定。

## 获取成员方法并调用

1. 1).public Method[] getMethods():获取所有"公有方法"；（包含了父类的方法也包含Object类）
2. 2).public Method[] getDeclaredMethods():获取所有的成员方法，包括私有的(不包括继承的)
3. 1).public Method getMethod(String name,Class<?>... parameterTypes):  参数： name : 方法名；  Class ... : 形参的Class类型对象
4. Method --> public Object invoke(Object obj,Object... args):
5. //2、获取main方法
6. Method methodMain = clazz.getMethod("main", String[].**class**);//第一个参数：方法名称，第二个参数：方法形参的类型，
7. //3、调用main方法
8. // methodMain.invoke(null, new String[]{"a","b","c"});
9. //第一个参数，对象类型，因为方法是static静态的，所以为null可以，第二个参数是String数组，这里要注意在jdk1.4时是数组，jdk1.5之后是可变参数
10. //这里拆的时候将  new String[]{"a","b","c"} 拆成3个对象。。。所以需要将它强转。
11. methodMain.invoke(**null**, (Object)**new** String[]{"a","b","c"});//方式一
12. // methodMain.invoke(null, new Object[]{new String[]{"a","b","c"}});//方式二

## 反射方法的其它使用之-通过反射运行配置文件内容

1. /\*
2. \* 我们利用反射和配置文件，可以使：应用程序更新时，对源码无需进行任何修改
3. \* 我们只需要将新类发送给客户端，并修改配置文件即可
4. \*/
5. **public** **class** Demo {
6. **public** **static** **void** main(String[] args) **throws** Exception {
7. //通过反射获取Class对象
8. Class stuClass = Class.forName(getValue("className"));//"cn.jxp.Student"
9. //2获取show()方法
10. Method m = stuClass.getMethod(getValue("methodName"));//show
11. //3.调用show()方法
12. m.invoke(stuClass.getConstructor().newInstance());
14. }
16. //此方法接收一个key，在配置文件中获取相应的value
17. **public** **static** String getValue(String key) **throws** IOException{
18. Properties pro = **new** Properties();//获取配置文件的对象
19. FileReader in = **new** FileReader("pro.txt");//获取输入流
20. pro.load(in);//将流加载到配置文件对象中
21. in.close();
22. **return** pro.getProperty(key);//返回根据key获取的value值
23. }
24. }

pro.properties存放

className = cn.fanshe.Student2

methodName = show2

## 反射方法的其它使用之-通过反射越过泛型检查

 泛型用在编译期，编译过后泛型擦除（消失掉）。所以是可以通过反射越过泛型检查的。

1. ArrayList<String> strList = new ArrayList<>();
2. strList.add("aaa");
3. strList.add("bbb");
5. //  strList.add(100);
6. //获取ArrayList的Class对象，反向的调用add()方法，添加数据
7. Class listClass = strList.getClass(); //得到 strList 对象的字节码 对象
8. //获取add()方法
9. Method m = listClass.getMethod("add", Object.class);
10. //调用add()方法
11. m.invoke(strList, 100);