作者：奋斗的小子  
链接：https://www.zhihu.com/question/24304289/answer/38218810  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

**反射之中包含了一个“反”的概念，所以要想解释反射就必须先从“正”开始解释，一般而言，当用户使用一个类的时候，应该先知道这个类，而后通过这个类产生实例化对象，但是“反”指的是通过对象找到类。**

packagecn.mldn.demo;

classPerson {}

publicclassTestDemo {

publicstaticvoidmain(String[] args) throwsException {

Person per = newPerson() ; // 正着操作

System.out.println(per.getClass().getName()); // 反着来

}

}

**以上的代码使用了一个getClass()方法，而后就可以得到对象所在的“包.类”名称，这就属于“反”了，但是在这个“反”的操作之中有一个getClass()就作为发起一切反射操作的开端。**

**Person的父类是Object类，而上面所使用getClass()方法就是Object类之中所定义的方法。**

**·取得Class对象：public final Class<?> getClass()，反射之中的所有泛型都定义为?，返回值都是Object。**

**而这个getClass()方法返回的对象是Class类的对象，所以这个Class就是所有反射操作的源头。但是在讲解其真正使用之前还有一个需要先解释的问题，既然Class是所有反射操作的源头，那么这个类肯定是最为重要的，而如果要想取得这个类的实例化对象，Java中定义了三种方式：**

**方式一：通过Object类的getClass()方法取得，基本不用：**

packagecn.mldn.demo;

classPerson {}

publicclassTestDemo {

publicstaticvoidmain(String[] args) throwsException {

Person per = newPerson() ; // 正着操作

Class<?> cls = per.getClass() ; // 取得Class对象

System.out.println(cls.getName()); // 反着来

}

}

**方式二：使用“类.class”取得，在日后学习Hibernate开发的时候使用**

packagecn.mldn.demo;

classPerson {}

publicclassTestDemo {

publicstaticvoidmain(String[] args) throwsException {

Class<?> cls = Person.class; // 取得Class对象

System.out.println(cls.getName()); // 反着来

}

}

**方式三：使用Class类内部定义的一个static方法，主要使用**

**·取得Class类对象：public static Class<?> forName(String className) throws ClassNotFoundException；**

packagecn.mldn.demo;

classPerson {}

publicclassTestDemo {

publicstaticvoidmain(String[] args) throwsException {

Class<?> cls = Class.forName("cn.mldn.demo.Person") ; // 取得Class对象

System.out.println(cls.getName()); // 反着来

}

}

**那么现在一个新的问题又来了，取得了Class类的对象有什么用处呢？对于对象的实例化操作之前一直依靠构造方法和关键字new完成，可是有了Class类对象之后，现在又提供了另外一种对象的实例化方法：**

**·通过反射实例化对象：public T newInstance() throws InstantiationException, IllegalAccessException；**

**范例：通过反射实例化对象**

packagecn.mldn.demo;

classPerson {

@Override

publicString toString() {

return"Person Class Instance .";

}

}

publicclassTestDemo {

publicstaticvoidmain(String[] args) throwsException {

Class<?> cls = Class.forName("cn.mldn.demo.Person") ; // 取得Class对象

Object obj = cls.newInstance() ; // 实例化对象，和使用关键字new一样

Person per = (Person) obj ; // 向下转型

System.out.println(per);

}

}

**那么现在可以发现，对于对象的实例化操作，除了使用关键字new之外又多了一个反射机制操作，而且这个操作要比之前使用的new复杂一些，可是有什么用？**

**对于程序的开发模式之前一直强调：尽量减少耦合，而减少耦合的最好做法是使用接口，但是就算使用了接口也逃不出关键字new，所以实际上new是造成耦合的关键元凶。**

**范例：回顾一下之前所编写的工厂设计模式**

packagecn.mldn.demo;

interfaceFruit {

publicvoideat() ;

}

classApple implementsFruit {

publicvoideat() {

System.out.println("吃苹果。");

};

}

classFactory {

publicstaticFruit getInstance(String className) {

if("apple".equals(className)){

returnnewApple() ;

}

returnnull;

}

}

publicclassFactoryDemo {

publicstaticvoidmain(String[] args) {

Fruit f = Factory.getInstance("apple") ;

f.eat() ;

}

}

**以上为之前所编写最简单的工厂设计模式，但是在这个工厂设计模式之中有一个最大的问题：如果现在接口的子类增加了，那么工厂类肯定需要修改，这是它所面临的最大问题，而这个最大问题造成的关键性的病因是new，那么如果说现在不使用关键字new了，变为了反射机制呢？**

**反射机制实例化对象的时候实际上只需要“包.类”就可以，于是根据此操作，修改工厂设计模式。**

packagecn.mldn.demo;

interfaceFruit {

publicvoideat() ;

}

classApple implementsFruit {

publicvoideat() {

System.out.println("吃苹果。");

};

}

classOrange implementsFruit {

publicvoideat() {

System.out.println("吃橘子。");

};

}

classFactory {

publicstaticFruit getInstance(String className) {

Fruit f = null;

try{

f = (Fruit) Class.forName(className).newInstance() ;

} catch(Exception e) {

e.printStackTrace();

}

returnf ;

}

}

publicclassFactoryDemo {

publicstaticvoidmain(String[] args) {

Fruit f = Factory.getInstance("cn.mldn.demo.Orange") ;

f.eat() ;

}

}

**发现，这个时候即使增加了接口的子类，工厂类照样可以完成对象的实例化操作，这个才是真正的工厂类，可以应对于所有的变化。如果单独从开发角度而言，与开发者关系不大，但是对于日后学习的一些框架技术这个就是它实现的命脉，在日后的程序开发上，如果发现操作的过程之中需要传递了一个完整的“包.类”名称的时候几乎都是反射机制作用。**

3.12.2 、反射的深入应用

**以上只是利用了Class类作为了反射实例化对象的基本应用，但是对于一个实例化对象而言，它需要调用类之中的构造方法、普通方法、属性，而这些操作都可以通过反射机制完成。**

3.12.2 .1、调用构造

**使用反射机制也可以取得类之中的构造方法，这个方法在Class类之中已经明确定义了：**

**以下两个方法**

**取得一个类的全部构造：**

**public Constructor<?>[] getConstructors() throws SecurityException**

**取得一个类的指定参数构造：**

**public Constructor<T> getConstructor(Class<?>... parameterTypes) throws NoSuchMethodException, SecurityException**

**现在发现以上的两个方法返回的都是java.lang.reflect.Constructor类的对象。**

**范例：取得一个类之中的全部构造**

packagecn.mldn.demo;

importjava.lang.reflect.Constructor;

classPerson { // CTRL + K

publicPerson() {}

publicPerson(String name) {}

publicPerson(String name,intage) {}

}

publicclassTestDemo {

publicstaticvoidmain(String[] args) throwsException {

Class<?> cls = Class.forName("cn.mldn.demo.Person") ; // 取得Class对象

Constructor<?> cons [] = cls.getConstructors() ; // 取得全部构造

for(intx = 0; x < cons.length; x++) {

System.out.println(cons[x]);

}

}

}

**验证：在之前强调的一个简单Java类必须存在一个无参构造方法**

**范例：观察没有无参构造的情况**

packagecn.mldn.demo;

classPerson { // CTRL + K

privateString name;

privateintage;

publicPerson(String name,intage) {

this.name= name ;

this.age= age ;

}

@Override

publicString toString() {

return"Person [name="+ name+ ", age="+ age+ "]";

}

}

publicclassTestDemo {

publicstaticvoidmain(String[] args) throwsException {

Class<?> cls = Class.forName("cn.mldn.demo.Person") ; // 取得Class对象

Object obj = cls.newInstance(); // 实例化对象

System.out.println(obj);

}

}

**此时程序运行的时候出现了错误提示“java.lang.InstantiationException”，因为以上的方式使用反射实例化对象时需要的是类之中要提供无参构造方法，但是现在既然没有了无参构造方法，那么就必须明确的找到一个构造方法，而后利用Constructor类之中的新方法实例化对象：**

**·实例化对象：public T newInstance(Object... initargs) throws InstantiationException, IllegalAccessException,IllegalArgumentException, InvocationTargetException**

packagecn.mldn.demo;

importjava.lang.reflect.Constructor;

classPerson { // CTRL + K

privateString name;

privateintage;

publicPerson(String name,intage) {

this.name= name ;

this.age= age ;

}

@Override

publicString toString() {

return"Person [name="+ name+ ", age="+ age+ "]";

}

}

publicclassTestDemo {

publicstaticvoidmain(String[] args) throwsException {

Class<?> cls = Class.forName("cn.mldn.demo.Person") ; // 取得Class对象

// 取得指定参数类型的构造方法

Constructor<?> cons = cls.getConstructor(String.class,int.class) ;

Object obj = cons.newInstance("张三", 20); // 为构造方法传递参数

System.out.println(obj);

}

}

**很明显，调用无参构造方法实例化对象要比调用有参构造的更加简单、方便，所以在日后的所有开发之中，凡是有简单Java类出现的地方，都一定要提供无参构造。**

3.12.2 .2、调用普通方法

**当取得了一个类实例化对象之后，下面最需要调用的肯定是类之中的方法，所以可以继续使用Class类取得一个类中所定义的方法定义：**

**·取得全部方法：public Method[] getMethods() throws SecurityException；**

**·取得指定方法：public Method getMethod(String name, Class<?>... parameterTypes) throws NoSuchMethodException, SecurityException**

**发现以上的方法返回的都是java.lang.reflect.Method类的对象。**

**范例：取得一个类之中所定义的全部方法**

packagecn.mldn.demo;

importjava.lang.reflect.Method;

classPerson {

privateString name;

publicvoidsetName(String name) {

this.name= name;

}

publicString getName() {

returnname;

}

}

publicclassTestDemo {

publicstaticvoidmain(String[] args) throwsException {

Class<?> cls = Class.forName("cn.mldn.demo.Person") ; // 取得Class对象

Method met [] = cls.getMethods() ; // 取得全部方法

for(intx = 0; x < met.length; x++) {

System.out.println(met[x]);

}

}

}

**但是取得了Method类对象最大的作用不再于方法的列出（方法的列出都在开发工具上使用了），但是对于取得了Method类对象之后还有一个最大的功能，就是可以利用反射调用类中的方法：**

**·调用方法：public Object invoke(Object obj, Object... args) throws IllegalAccessException,IllegalArgumentException, InvocationTargetException**

**之前调用类中方法的时候使用的都是“对象.方法”，但是现在有了反射之后，可以直接利用Object类调用指定子类的操作方法。（同时解释一下，为什么setter、getter方法的命名要求如此严格）。**

**范例：利用反射调用Person类之中的setName()、getName()方法**

packagecn.mldn.demo;

importjava.lang.reflect.Method;

classPerson {

privateString name;

publicvoidsetName(String name) {

this.name= name;

}

publicString getName() {

returnname;

}

}

publicclassTestDemo {

publicstaticvoidmain(String[] args) throwsException {

Class<?> cls = Class.forName("cn.mldn.demo.Person") ; // 取得Class对象

Object obj = cls.newInstance(); // 实例化对象，没有向Person转型

String attribute = "name"; // 要调用类之中的属性

Method setMet = cls.getMethod("set"+ initcap(attribute), String.class);// setName()

Method getMet = cls.getMethod("get"+ initcap(attribute));// getName()

setMet.invoke(obj, "张三") ; // 等价于：Person对象.setName("张三")

System.out.println(getMet.invoke(obj));// 等价于：Person对象.getName()

}

publicstaticString initcap(String str) {

returnstr.substring(0,1).toUpperCase().concat(str.substring(1)) ;

}

}

**在日后的所有框架技术开发之中，简单Java类都是如此应用的，所以必须按照标准进行。**

3.12.2 .3、调用成员

**类之中最后一个组成部分就是成员（Field，也可以称为属性），如果要通过反射取得类的成员可以使用方法如下：**

**·取得本类的全部成员：public Field[] getDeclaredFields() throws SecurityException；**

**·取得指定的成员：public Field getDeclaredField(String name) throws NoSuchFieldException, SecurityException；**

**这两个方法的返回值类型是java.lang.reflect.Field类的对象，下面首先观察如何取得一个类之中的全部属性。**

**范例：取得一个类之中的全部属性**

packagecn.mldn.demo;

importjava.lang.reflect.Field;

classPerson {

privateString name;

}

publicclassTestDemo {

publicstaticvoidmain(String[] args) throwsException {

Class<?> cls = Class.forName("cn.mldn.demo.Person") ; // 取得Class对象

Field field [] = cls.getDeclaredFields() ; // 取得全部属性

for(intx = 0; x < field.length; x++) {

System.out.println(field[x]);

}

}

}

**但是找到Field实际上就找到了一个很有意思的操作，在Field类之中提供了两个方法：**

**·设置属性内容（类似于：对象.属性= 内容）：public void set(Object obj, Object value)**

**throws IllegalArgumentException, IllegalAccessException；**

**·取得属性内容（类似于：对象.属性）：public Object get(Object obj)**

**throws IllegalArgumentException, IllegalAccessException**

**可是从类的开发要求而言，一直都强调类之中的属性必须封装，所以现在调用之前要想办法解除封装。**

**·解除封装：public void setAccessible(boolean flag) throws SecurityException；**

**范例：利用反射操作类中的属性**

packagecn.mldn.demo;

importjava.lang.reflect.Field;

classPerson {

privateString name;

}

publicclassTestDemo {

publicstaticvoidmain(String[] args) throwsException {

Class<?> cls = Class.forName("cn.mldn.demo.Person"); // 取得Class对象

Object obj = cls.newInstance(); // 对象实例化属性才会分配空间

Field nameField = cls.getDeclaredField("name") ; // 找到name属性

nameField.setAccessible(true) ; // 解除封装了

nameField.set(obj, "张三") ; // Person对象.name = "张三"

System.out.println(nameField.get(obj)); // Person对象.name

}

}