# 反射

JAVA反射机制是在运行状态中，对于任意一个类，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的方法的功能称为java语言的反射机制。

blog：https://www.cnblogs.com/nerxious/archive/2012/12/24/2829446.html

# 获取对象的运行时对象的类

object.getClass(): 获取对象的运行时对象的类

返回: class packagename/classname

# 类名

object.getClass().getName(): 获取类名

返回: packagename/classname

# Class类的实例化

由于Class类没有构造方法，所以实例化Class类的方式有点特殊，有三种方式：

1. 对象.getClass( )

2. 类.Class

3. forName()

forName()可以在类不确定的情况下实例化Class，更具灵活性

# 创建对象实例

Class类中有一个方法叫做newInstance( ),它可以用来创建一个Class类对象的新实例

# Class类的有参构造对象

构造方法需要使用getConstructor( )方法获得，参数类型则是：原有类型.class

无论是有参还是无参，这里所使用的构造方法，原本的类里面必须对应存在

# 获取类的构造方法

使用getConstructors()方法获取，该方法返回的是Constructor数组

getDeclaredConstructors()的返回所有类型的构造器，包括public和非public

getConstructors()只返回public。

# 取得类所实现的接口

使用getInterfaces()获取一个类所实现的接口，该方法返回的是一个Class数组

# 取得父类对象

getSuperclass():返回一个类的父类对象，java中是单继承，父类只有一个

# 取得类的全部方法

getDeclaredMethod：获取当前类的所有声明的方法，包括public、protected和private修饰的方法。需要注意的是，这些方法一定是在当前类中声明的，从父类中继承的不算，实现接口的方法由于有声明所以包括在内。

getMethod：获取当前类和父类的所有public的方法。这里的父类，指的是继承层次中的所有父类。比如说，A继承B，B继承C，那么B和C都属于A的父类。

# 取得本类的全部属性

getFields()：获得某个类的所有的公共（public）的字段，包括父类中的字段。

getDeclaredFields()：获得某个类的所有声明的字段，即包括public、private和proteced，但是不包括父类的申明字段。

# 获取本类中属性的值

获取公共属性的值:

Field f1 = c.getField("name");

String str = (String) f1.get(p); // get(p)表明要获取是哪个对象的值

获取私有属性的值:

Field f2 = c.getDeclaredField("age");

f2.setAccessible(true); // age是私有属性，所以要设置安全检查为true  
int age = (int) f2.get(p);

通过反射修改属性

Field f = c.getDeclaredField("name");

f.setAccessible(true);  
f.set(p, "张二蛋"); // 修改属性，传入要设置的对象和值

# 通过反射调用方法

//getMethod()方法需要传入方法名，和参数类型

Method m1 = c.getMethod("print", int.class);   
//invoke()表示调用的意思，需要传入对象和参数   
m1.invoke(p, 10);

# 通过反射操作数组

getComponentType( )返回的是数组元素的Class

# JAVA 反射修改static,final修饰的变量

调用对应Class的getDeclaredField或getField方法，获取要修改的Filed；

2个方法的差别在于：

getDeclaredField可获取当前Class内所有变量名（private，protect，public，friend），但不会获取父类变量；

getField可获取当前Class及父Class内所有访问级别为public的变量名；

final修饰的常量不可修改，判断field对应数据为常量则直接返回false；

常量的判断条件：

1）使用了final修饰

2）数据类型为基本类型或者String类型

原因：

使用final修饰后，被引用的数据地址将不可改变，我们只能尝试修改地址上的内容，而常量不能修改地址内容，或者说修改不生效。

如果访问级别不是public，调用setAccessible(true)获得访问权限；

如果使用了final修饰，而没有使用static修饰，可以调用setAccessible(true)获得修改权限，或者修改Modifier，去除final修饰符；

如果同时使用了static和final，则只能通过修改Modifier去除final修饰符来获取修改权限；

判断要修改的数据类型，如果为基本类型，调用对应的基本类型修改方法，其他情况直接调用set方法；

对修改过的部分还原。

# JAVA中类、实例与Class对象

## 类

类是面向对象编程语言的一个重要概念，它是对一项事物的抽象概括，可以包含该事物的一些属性定义，以及操作属性的方法。面向对象编程中，我们都是以类来编码。

## 实例

简单理解，就是new，就是对类的实例化，创建这个类对应的实际对象，类只是对事物的描述，而实例化就相当于为这个描述新开辟了一块内存，可以改变这块区域里的各种属性（成员变量），当然，也可以实例化多块区域，只是不同的对象而已。

## Class

注意这里C大写了，与类概念区分开，在java里，Class是一个实实在在的类，在包 java.lang 下，有这样一个Class.java文件，它跟我们自己定义的类一样，是一个实实在在的类，Class对象就是这个Class类的实例了。在Java里，所有的类的根源都是Object类，而Class也不例外，它是继承自Object的一个特殊的类，它内部可以记录类的成员、接口等信息，也就是在Java里，Class是一个用来表示类的类。（o(∩\_∩)o 有点绕啊，抓住关键一点，Class是一个实实在在的类，可以为它创建实例，也就是本文后面提到的Class对象，也看叫做Class实例）。

java提供了下面几种获取到类的Class对象的方法：

1) 利用对象实例调用getClass()方法获取该对象的Class实例；

2) 使用Class类的静态方法forName("包名+类名")，用类的名字获取一个Class实例

3)运用 类名.class 的方式来获取Class实例；

我们知道java世界是运行在JVM之上的，我们编写的类代码，在经过编译器编译之后，会为每个类生成对应的.class文件，这个就是JVM可以加载执行的字节码。运行时期间，当我们需要实例化任何一个类时，JVM会首先尝试看看在内存中是否有这个类，如果有，那么会直接创建类实例；如果没有，那么就会根据类名去加载这个类，当加载一个类，或者当加载器(class loader)的defineClass()被JVM调用，便会为这个类产生一个Class对象（一个Class类的实例），用来表达这个类，该类的所有实例都共同拥有着这个Class对象，而且是唯一的。

## 总结

在java里，类只是信息描述的，写明了有哪些内部属性及接口，你可以理解为是定义了一套规则；而Class对象在java里被用来对类的情况进行表述的一个实例，也就是是类的实际表征，可以理解为是对规则的图表化，这样JVM才能直观的看懂，可以看做是一个模版；而类的实例化对象，就是通过模版，开辟出的一块内存进行实际的使用。

# 目录1

## 目录2

### 目录3

#### 目录4

##### 目录5