# 什么是反射？

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（包括私有方法和属性）；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（注意关键词：运行状态）换句话说，Java程序可以加载一个运行时才得知名称的class，获悉其完整构造（但不包括methods定义），并生成其对象实体、或对其fields设值、或唤起其methods。

主要是五个类：Class、Method、Field、Constractor和Array

# Class：

在程序运行期间，Java运行时系统始终为所有的对象维护一个被称为运行时的类型标识。这个信息跟踪着每个对象所属的类。JVM利用运行时信息选择相应的方法执行。而保存这些信息的类称为Class。可能容易产生混淆，容易想到class。不过二者没什么关系，class不过是描述类的一个关键字。而Class却是保存着运行时信息的类。

Class是反射的起源也是入口，唯有获取到Class对象才能进行后续反射操作。那么如何获取这个对象呢，查看源码可以得知Class类没有公共构造方法，也就是说不能通过显示方法构造Class对象，而是由jvm在第一次加载类时由类加载器生成。因此我们通常只用关心Class对象的获取和使用即可。Class对象获取主要有三种方式

1. Class.forName(“完整类名”)
2. 实例对象.getClass()
3. 类名.class

三种方式常用第三种，第一种对象都有了还要反射干什么。第二种需要导入类的包，依赖太强，不导包就抛编译错误。一般都第三种，一个字符串可以传入也可写在配置文件中等多种方法。

Class主要有以下方法可供调用：

forName() (1)获取Class对象的一个引用，但引用的类还没有加载(该类的第一个对象没有生成)就加载了这个类。(2)为了产生Class引用，forName()立即就进行了初始化

Object.getClass() 获取Class对象的一个引用

getName() 获取Class对象的全限定类名，也就是包名+类名

getSimpleName 单纯获取Class对象类名

isInterface() 判断类是否接口

getInterfaces() 获取类的全部实现接口

getSuperClass（） 获取类的直接基类，也就是父类

newInstance（） 生成类的无参构造函数生成类对象，使用该方法时，类必须带有无参构造器，且不可为私有

getDeclaredMethod（String name,Class[] parms） 返回对象Method，获取类中自己声明的所有方法，通过此方法可获取类中所有声明方法.类似还getDeclaredField（）,getConstructor（）

getMethods（） 返回对象Array 获取类（包含父类）中的所有公共方法，类似还有getFields（），getConstructor（）。

# Method：

Method是一个类,位于java.lang.reflect包下，继承自Executable类。在Java反射中 Method类描述的是 类的方法信息。Method的获取方法都在前面介绍的Class类中,主要方法：

方法名 说明

getName（） 获取方法名

getReturnType() 返回Class对象，表示返回值类型

getParameters() 返回Parameter数组，表示参数序列

getTypeParameters() 返回一个TypeVariable对象数组，表示该方法对象声明列表上的类型变量数组

invole(Object obj,Objece[] parms) 调用method方法，第一个为调用对象，后面是方法参数值（private 方法设置accessible为true时才可正常访问，否则会报错）

isAccessible() 返回对象可访问标识

setAccessible() 设置可访问标识，设置为true时，可访问对象的private方法

getAnnotations（） 返回方法上的所有注解

getAnnotationByType(Class clz) 如果该方法对象存在指定类型的注解，则返回该注解数组，否则返回null

# Field：

Field类同样位于java.lang.reflect包下，为我们提供了获取当前对象成员变量类型和重新设值的方法。获取方式同样在Class类中。主要方法：

方法 说明

getType（） 获取当前变量类型：基本类型或引用类型

getModifier() 获取当前变量修饰符

setAccessible() 设置可访问标识

set（Object obj,Object value） 为对象设置当前属性值，类似方法还有setInt(),setShort()…

# Constructor:

类的构造方法，继承自Executable类，同样在Class中通过getConstructors()和getDeclaredConstructors()获得，特定构造方法可通过getDeclaredConstructor（Class[] parms）获取，parms代表参数类型。借助Constructor可直接生成类实例。

方法 说明

newInstance(Object … initargs) 返回对应构造方法对象，参数为可变参数

setAccessible() 设置可访问标识

getAnnotations() 获取构造方法上的所有注解

# Array:

Array提供了动态创建和访问数组元素的各种静态方法，位于java.lang.reflect包下。注意和Arrays进行区分，Arrays主要对数组进行复制，排序、搜索等操作；而Array主要供反射调用，动态创建数组，并进行访问和修改，用于反射时参数的读取和录入。主要方法：

方法 说明

newInstance（Class clz,int length） 构建clz类型，容量为length的数组返回类型为Object

get（Object obj,int index） 访问obj数组的index下标的对应元素,类似的还有getBoolean（）等…

getLength（） 返回数组容量（注意是总容量，不是当前长度）

set(Object array,Object item,int index) 设置array中index对应值为item，同样还有setInt（）等…

# 反射在Android中的应用

# 通过反射获取构造方法为私有的类，而不需要进行导包操作

Person类：

public class Person {

String name;

int age;

private Person() {

}

private Person(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

@Override

public String toString() {

return "Person{" +

"name='" + name + '\'' +

", age=" + age +

'}';

}

}

可以看到Person类只提供了私有构造方法，因此通过常规方式，我们无法从外部得到一个Person对象，而借助反射我们可以生成一个Person类的实例，还能进行成员方法的调用，并且是发生在不导包的情况下。

public static void main(String[] args) throws ClassNotFoundException, InstantiationException, IllegalAccessException {

//通过完整类名获取Class对象

Class c = Class.forName("com.example.beans.Person");

try {

//获取构造函数对象

Constructor constructor=c.getDeclaredConstructor(String.class,int.class);

//由于构造方法是私有的因此需要将可访问标识设为true

constructor.setAccessible(true);

//通过构造函数生成Person实例

Object bean = constructor.newInstance("张三", 32);

//输出Person{name='张三', age=32}

System.out.println(bean.toString())

//获取无参构造函数对象

Constructor constructor1=c.getDeclaredConstructor();

constructor1.setAccessible(true);

//构造无参实例

Object bean2 = constructor1.newInstance();

//获取setName方法对象

Method setName = c.getDeclaredMethod("setName", String.class);

//通过setName方法设置name值

setName.invoke(bean2, "李四");

//获取setAge方法

Method setAge = c.getDeclaredMethod("setAge", int.class);

//设置age值

setAge.invoke(bean2, 45);

//输出Person{name='李四', age=45}

System.out.println(bean2.toString());

//获取name属性对象

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

//设置name属性可访问

nameField.setAccessible(true);

//设置name值

nameField.set(bean2, "王五");

//输出Person{name='王五', age=45}

System.out.println(bean2.toString());

} catch (NoSuchMethodException e) {

e.printStackTrace();

} catch (InvocationTargetException e) {

e.printStackTrace();

} catch (NoSuchFieldException e) {

e.printStackTrace();

}

}

通过上面代码，我们借助类名获取了Person的Class类对象c，并借助此对象进一步获取到了构造方法、成员方法、成员属性等一系列对象，从而进行Person实例的构造以及方法的调用和属性的设置等操作。

# 全局View点击监听

试想某天产品提出了全局防止多次频繁的需求，界面稍微多点，逐个设置监听就是一场灾难。然而借助反射，我们可以轻松实现，只需一行代码即可搞定一个Activity（如果有BaseActivity的话就更方便了）。话不多少，先贴代码：

package com.example.reflectpractice;

import android.util.Log;

import android.view.View;

import android.view.ViewGroup;

import java.lang.reflect.Field;

import java.lang.reflect.InvocationTargetException;

import java.lang.reflect.Method;

/\*\*

\* @User Created By Mr.m

\* @Date 2019/8/2

\*\*/

public class HookClickListenerUtils {

/\*\*

\*通过递归对每子View执行hook函数

\*\*/

public static void hookViewGroup(View view) {

if (view instanceof ViewGroup) {

//遍历子View

for (int i = 0; i < ((ViewGroup) view).getChildCount(); i++) {

if (((ViewGroup) view).getChildAt(i) instanceof ViewGroup) {

//递归调用

hockViewGroup(((ViewGroup) view).getChildAt(i));

} else {

hockView(((ViewGroup) view).getChildAt(i));

}

}

} else {

hockView(view);

}

}

/\*\*

\*对View进行hook

\*\*/

public static void hookView(View view) {

try {

//获取View的Class对象

Class cls = Class.forName("android.view.View");

//获取getListenerInfo方法

Method getListenerInfoMethod = cls.getDeclaredMethod("getListenerInfo");

//方法不可访问时设置可访问标识

if (!getListenerInfoMethod.isAccessible()) {

getListenerInfoMethod.setAccessible(true);

}

//获取具体View的ListenerInfo对象

Object listenerInfoObject = getListenerInfoMethod.invoke(view);

//获取ListenerInfo内部类对象

Class listenerInfoCls = Class.forName("android.view.View$ListenerInfo");

//获取onClickListener属性

Field onClickListerField=listenerInfoCls

.getDeclaredField("mOnClickListener");

//设置onClickListener属性可访问

onClickListerField.setAccessible(true);

//为listenerInfo对象设置代理监听

onClickListerField.set(listenerInfoObject,new HookListener((View.OnClickListener)onClickListerField.get(listenerInfoObject)) );

} catch (ClassNotFoundException e) {

e.printStackTrace();

} catch (NoSuchMethodException e) {

e.printStackTrace();

} catch (IllegalAccessException e) {

e.printStackTrace();

} catch (InvocationTargetException e) {

e.printStackTrace();

} catch (NoSuchFieldException e) {

e.printStackTrace();

}

}

/\*\*

\*View 代理监听接口

\*\*/

private static class HookListener implements View.OnClickListener {

//View原监听对象

private View.OnClickListener onClickListener;

//两次点击最少间隔时间

private int MIN\_CLICK\_DELAY\_TIME = 3000;

//上次点击时间

private long lastClickTime = 0;

@Override

public void onClick(View view) {

//当前时间

long curTime = System.currentTimeMillis();

//判断是否大于最小间隔时间

if (curTime - lastClickTime > MIN\_CLICK\_DELAY\_TIME) {

onClickListener.onClick(view);

lastClickTime = curTime;

}else{

Log.i("频繁点击",""+view.getTag());

}

}

HookListener(View.OnClickListener listener) {

this.onClickListener = listener;

}

}

}

HookListenerUtils主要做了三件事：

对ViewGroup进行递归调用hookView（）方法

通过hookView（）方法，借助反射为View的onClickListener设置代理对象

定义HookListener静态内部类，用作代理对象，执行点击时的具体策略，唤起view的onClick还是另作频繁点击处理。

通过调用HookViewGroup（）将对应的ViewGroup作为参数传入即可实现所有子View的频繁点击处理，因此我们自然而然的想到使用DecorView来进行当前Activity下所有View的处理，如此全局监听也就完成了。

# 调用其他jar包或者apk中的方法

略

https://blog.csdn.net/qq\_36401386/article/details/100584490?ops\_request\_misc=%257B%2522request%255Fid%2522%253A%2522169113349016800185879784%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request\_id=169113349016800185879784&biz\_id=0&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~sobaiduend~default-2-100584490-null-null.142^v92^chatsearchT0\_1&utm\_term=%E5%AE%89%E5%8D%93%E5%8F%8D%E5%B0%84&spm=1018.2226.3001.4187

# 通过反射运行配置文件内容

可以将要运行的类写入配置文件中动态加载进来，当要改变类时只需要改变配置文件中的内容即可。

# 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