Java反射-高级开发必须懂的

rocomp [Java架构师之路](javascript:void(0);) 昨天

作者： rocomp

<https://www.cnblogs.com/rocomp/p/4781987.html>

Java架构师之路做了编排

理解反射对学习Java框架有很大的帮助，如Spring框架的核心就是使用Java反射实现的，而且对做一些Java底层的操作会很有帮助。

# 一：Class类的使用

①.万事万物皆对象，（当然，基本数据类型，静态成员不是面向对象（属于类的）），所以我们创建的每一个类也都是对象，即类本身是java.lang.Class类的实例对象，但是这些对象都不需要new出来，因为java.lang.Class类的构造方法是私有的。

②.任何一个类都是Class类的实例对象，这个实例对象有三种表示方式：(我们新建一个Student类)

Class c1 = Student.class;//实际告诉我们任何一个类都有一个隐含的静态成员变量class（知道类名时用）  
Class c2 = stu.getClass();//已知该类的对象通过getClass方法（知道对象时用）    
Class c3 = Class.forName("类的全名");//会有一个ClassNotFoundException异常

官网解释说：c1,c2表示了Student类的类类型（）class type)，万事万物皆对象，类也是对象，是Class类的实例对象，这个对象我们成为该类的类类型（有点乱，但是慢慢捋一下还是能理解的）

这里有一点值得注意，当我们执行System.out.println(c1==c2);语句，结果返回的是true，这是为什么呢？原因是不管c1还是c2都代表了Student类的类类型，一个类可能是Class类的一个实例对象。

我们完全可以通过类的类类型创建该类的对象实例，即通过c1或c2创建Student的实例。

Student stu = (Student)c1.newInstance();//前提是必须要有无参的构造方法，因为该语句会去调用其无参构造方法。该语句会抛出异常。

# 二：动态加载类

**①.编译时加载类是静态加载类**

new 创建对象是静态加载类，在编译时刻就需要加载所有可用使用到的类，如果有一个用不了，那么整个文件都无法通过编译

**②.运行时加载类是动态加载类**

    Class c =  Class.forName("类的全名")，不仅表示了类的类型，还表示了动态加载类，编译不会报错，在运行时才会加载，使用接口标准能更方便动态加载类的实现。功能性的类尽量使用动态加载，而不用静态加载。

    很多软件比如QQ,360的在线升级，并不需要重新编译文件，只是动态的加载新的东西

# 三：获取方法信息

①基本的数据类型，void关键字都存在类类型

Class c1 =int.class;//int的类类型  
Class c2 =String.class;//String类的类类型，可以理解为编译生成的那个String.class字节码文件，  
//当然，这并不是官方的说法  
Class c3 =double.class;  
Class c4 =Double.class;  
Class c5 =void.class;

②Class类的基本API操作

 /\*\*  
 \* 打印类的信息，包括类的成员函数，成员变量  
 \* @param obj 该对象所属类的信息  
 \*/  
publicstaticvoid printClassMessage(Object obj){  
    //要获取类的信息，首先要获取类的类类型  
    Class c = obj.getClass();//传递的是哪个子类的对象，c就是该子类的类类型  
    //获取类的名称  
    System.out.println("累的名称是："+c.getName());  
  
    /\*  
     \* Method类，方法的对象  
     \* 一个成员方法就是一个Method对象  
     \* getMethods()方法获取的是所有的public的函数，包括父类继承而来的  
     \* getDeclaredMethods()获取的是多有该类自己声明的方法，不问访问权限  
     \*/  
    Method[] ms = c.getMethods();//c.getDeclaredMethods();  
    for(int i =0; i < ms.length; i++){  
        //得到方法的返回值类型的类类型  
        Class retrunType = ms[i].getReturnType();  
        System.out.print(retrunType.getName()+" ");  
        //得到方法的名称  
        System.out.print(ms[i].getName()+"(");  
        //获取的参数类型--->得到的是参数列表的类型的类类型  
        Class[] paraTypes = ms[i].getParameterTypes();  
        for(Class class1 : paraTypes){  
            System.out.print(class1.getName()+",");  
        }  
        System.out.println(")");  
    }  
}

Class的API中还有很多其他的方法，可以得到interface、Package、Annotation等很多信息，具体使用请参考帮助手册，本文就不在详细讲解。特别注意的一点是，如果你想得到一个类的信息，首先就要获取该类的类类型。

# 四：获取成员变量构造函数信息

/\*\*  
 \* 成员变量也是对象，是java.lang.reflect.Field这个类的的对象  
 \* Field类封装了关于成员变量的操作  
 \* getFields()方法获取的是所有public的成员变量的信息  
 \* getDeclareFields()方法获取的是该类自己声明的成员变量的信息  
 \*/  
Field[] fs = c.getDeclaredFields();  
for(Field field : fs){  
    //得到成员变量的类型的类类型  
    Class fieldType = field.getType();  
    String typeName = fieldType.getName();  
    //得到成员变量的名称  
    String fieldName = field.getName();  
    System.out.print(typeName+" "+fieldName);  
}  
  
  
/\*\*  
 \* 构造函数也是对象  
 \* java.lang.Constructor中封装了构造函数的信息  
 \* getConstructor()方法获取所有的public的构造函数  
 \* getDeclaredConstructors得到所有的构造函数  
 \*/  
Constructor[] cs = c.getDeclaredConstructors();  
for(Constructor constructor : cs){  
    System.out.print(constructor.getName()+"(");  
    //获取构造函数的参数列表---》得到的是参数雷彪的类类型  
     Class[] paramTypes = constructor.getParameterTypes();  
     for(Class class1 : paramTypes){  
        System.out.print(class1.getName()+",");  
    }  
    System.out.println(")");  
}

# 五：方法反射的基本操作

1.如何获取某个方法

   方法的名称和方法的参数列表才能唯一决定某个方法

   Method m = c.getDeclaredMethod("方法名"，可变参数列表（参数类型.class）)

2.方法的反射操作

   m.invoke(对象，参数列表)

   方法如果没有返回值，返回null，如果有返回值返回Object类型，然后再强制类型转换为原函数的返回值类型

# 六：通过反射了解集合泛型的本质

ArrayList list1 =newArrayList();  
ArrayList<String> list2 =newArrayList<String>();  
  
Class c1 = list1.getClass();  
Class c2 = list2.getClass();  
  
System.out.println(c1==c2);//结果为true，为什么？？

结果分析：因为反射的操作都是编译之后的操作，也就是运行时的操作，c1==c2返回true，说明编译之后集合的泛型是去泛型化的。

       那么我们就可以理解为，Java集合中的泛型，是用于防止错误类型元素输入的，比如在list2中我们add一个int，add(10)就会编译报错，那么这个泛型就可以只在编译阶段有效，通过了编译阶段，泛型就不存在了。可以验证，我们绕过编译，用反射动态的在list2中add一个int是可以成功的，只是这时因为list2中存储了多个不同类型的数据（String型，和int型），就不能用for-each来遍历了，会抛出类型转换错误异常ClassCastException。

# 七：关于Java类加载器内容的详解

## 1类的加载

当程序要使用某个类时，如果该类还未被加载到内存中，则系统会通过加载，连接，初始化三步来实现对这个类进行初始化

·加载：

       就是指将class文件读入内存，并为之创建一个Class对象，任何类被使用时系统都会建立一个Class对象

·连接：

       验证：确保被加载类的正确性

       准备：负责为类的静态成员分配内存，并设置默认初始化值

       解析：将类中的符号引用替换为直接引用

·初始化：

        局部变量保存在栈区：必须手动初始化

        new 的对象保存在堆区：虚拟机会进行默认初始化，基本数据类型初始化值为0，引用类型初始化值为null

## 2.类加载的时机（只加载一次）

以下时机仅表示第一次的时候

① 创建类的实例的时候

② 访问类的静态变量的时候

③ 调用类的静态方法的时候

④ 使用反射方式来强制创建某个类或接口对应的java.lang.Class对象

⑤ 初始化某个类的子类的时候

⑥ 直接使用java.exe命令来运行某个主类

## 3.类加载器

负责将.class文件加载到内存中，并为之生成对应的Class对象

虽然我们在开发过程中不需要关心类加载机制，但是了解这个机制我们就能更好的理解程序的运行

**4.类加载器的组成**

①Bootstrap ClassLoader 根类加载器

也被称为引导类加载器，负责Java核心类的加载，比如System类，在JDK中JRE的lib目录下rt.jar文件中的类。

②Extension ClassLoader 扩展类加载器

负责JRE的扩展目录中jar包的加载，在JDK中JRE的lib目录下ext目录。

③System ClassLoader 系统类加载器

负责在JVM启动时加载来自java命令的class文件，以及classpath环境变量所指定的jar包和类路径，主要是我们开发者自己写的类。

往期推荐

【1】 [通俗理解乐观锁和悲观锁](https://mp.weixin.qq.com/s?__biz=MzI3NjU2ODA5Mg==&mid=2247484165&idx=1&sn=06ac4db23d325697afd8b9fd6a7a4733&scene=21#wechat_redirect)

【2】 [面试系列-HashMap和Hashtable的区别](https://mp.weixin.qq.com/s?__biz=MzI3NjU2ODA5Mg==&mid=2247484165&idx=2&sn=06359972c29a741bbbae788476ba6fda&scene=21#wechat_redirect)

【3】 [面试系列-线程池详解](https://mp.weixin.qq.com/s?__biz=MzI3NjU2ODA5Mg==&mid=2247484187&idx=2&sn=d0058d2d39ba820b840779e4256ce6d8&scene=21#wechat_redirect)

【4】 [一次邮件发送协议SMTP问题排查](https://mp.weixin.qq.com/s?__biz=MzI3NjU2ODA5Mg==&mid=2247484187&idx=1&sn=3986a0949a2b0d3d9696bc52feeb004c&scene=21#wechat_redirect)

【5】 [郁闷的时候看看这头驴！](https://mp.weixin.qq.com/s?__biz=MzI3NjU2ODA5Mg==&mid=2247484205&idx=1&sn=24a0ce0020e03c625b3a9977ffd9fa7d&scene=21#wechat_redirect)