java新手入门：Java反射机制(四） PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/145/2021_2022_java_E6_96_B0_E6_89_8B_c104_145120.htm 执行结果（例）：private transient LinkedList$Entry header.private transient int size.图5-10a：找出所有fields #004 System.out.println("G: " ff[i].toGenericString()). private transient java.util.LinkedList.java.util.LinkedList$Entry ??java.util.LinkedList.headerprivate transient int java.util.LinkedList.size图5-10b：找出所有fields。本例在for 循环内使用toGenericString()，省事。 找出class参用（导入）的所有classes没有直接可用的Reflection API可以为我们找出某个class参用的所有其它classes。要获得这项信息，必须做苦工，一步一脚印逐一记录。我们必须观察所有fields的类型、所有methods（包括constructors）的参数类型和回返类型，剔除重复，留下唯一。这正是为什么图5-2程序代码要为tName()指定一个hashtable（而非一个null）做为第二自变量的缘故：hashtable可为我们储存元素（本例为字符串），又保证不重复。 本文讨论至此，几乎可以还原一个class的原貌（唯有methods 和ctors的定义无法取得）。接下来讨论Reflection的另三个动态性质：(1) 运行时生成instances，(2) 执行期唤起methods，(3) 运行时改动fields。 运行时生成instances欲生成对象实体，在Reflection 动态机制中有两种作法，一个针对“无自变量ctor”，一个针对“带参数ctor”。图6是面对“无自变量ctor”的例子。如果欲调用的是“带参数ctor“就比较麻烦些，图7是个例子，其中不再调用Class的newInstance()，而

是调用Constructor 的newInstance()。图7首先准备一个Class[] 做为ctor的参数类型（本例指定为一个double和一个int），然后以此为自变量调用getConstructor()，获得一个专属ctor。接下来再准备一个Object[] 做为ctor实参值（本例指定3.14159和125），调用上述专属ctor的newInstance()。 #001 Class c = Class.forName("DynTest").#002 Object obj = null.#003 obj = c.newInstance(). //不带自变量#004 System.out.println(obj).图6：动态生成“ Class object 所对应之class ”的对象实体；无自变量。 #001 Class c = Class.forName("DynTest").#002 Class[] pTypes = new Class[] { double.class, int.class }.#003 Constructor ctor = c.getConstructor(pTypes).#004 //指定parameter list，便可获得特定之ctor#005#006 Object obj = null.#007 Object[] arg = new Object[] {3.14159, 125}. //自变量#008 obj = ctor.newInstance(arg).#009 System.out.println(obj).图7：动态生成“ Class object 对应之class ”的对象实体；自变量以Object[] 表示。 运行时调用methods这个动作和上述调用“ 带参数之ctor ”相当类似。首先准备一个Class[]做为ctor的参数类型（本例指定其中一个是String，另一个是Hashtable），然后以此为自变量调用getMethod()，获得特定的Method object。接下来准备一个Object[]放置自变量，然后调用上述所得之特定Method object的invoke()，如图8。知道为什么索取Method object时不需指定回返类型吗？因为method overloading机制要求signature（署名式）必须唯一，而回返类型并非signature的一个成份。换句话说，只要指定了method名称和参数列，就一定指出了一个独一无二的method。 #001 public String func(String s, Hashtable ht)#002 {#003 …System.out.println("func

invoked"). return s.#004 }#005 public static void main(String args[])#006 {#007 Class c = Class.forName("Test").#008 Class ptypes[] = new Class[2].#009 ptypes[0] = Class.forName("java.lang.String").#010 ptypes[1] = Class.forName("java.util.Hashtable").#011 Method m = c.getMethod("func",ptypes).#012 Test obj = new Test().#013 Object args[] = new Object[2].#014 arg[0] = new String("Hello,world").#015 arg[1] = null.#016 Object r = m.invoke(obj, arg).#017 Integer rval = (String)r.#018 System.out.println(rval).#019 }图8：动态唤起method 运行时变更fields内容与先前两个动作相比，“变更field内容”轻松多了，因为它不需要参数和自变量。首先调用Class的getField()并指定field名称。获得特定的Field object之后便可直接调用Field的get()和set()，如图9。 #001 public class Test {#002 public double d.#003#004 public static void main(String args[])#005 {#006 Class c = Class.forName("Test").#007 Field f = c.getField("d"). //指定field 名称#008 Test obj = new Test().#009 System.out.println("d= " (Double)f.get(obj)).#010 f.set(obj, 12.34).#011 System.out.println("d= " obj.d).#012 }#013 } 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com