**中软国际考题-笔试**

1. 单选题

1、下面程序的输出结果是什么? （C）

public class Example extends Date {

public static void main(String[] args) {

new Example().test();

}

public void test() {

System.out.println(super.getClass().getName());

}

}

A. Date B. Object C. Example D. 运行时抛出异常

2、假定Tester 类有如下test方法：

public int test(int p1, Integer p2)

以下哪段代码能正确地动态调用一个Tester 对象的test方法？(C)

A. Class classType=Tester.class;

Object tester=classType.newInstance();

Method addMethod=classType.getMethod("test",new Class[]{int.class,int.class});

Object result=addMethod.invoke(tester,

new Object[]{new Integer(100),new Integer(200)});

B. Class classType=Tester.class;

Object tester=classType.newInstance();

Method addMethod=classType.getMethod("test",new Class[]{int.class,int.class});

int result=addMethod.invoke(tester,

new Object[]{new Integer(100),new Integer(200)});

C. Class classType=Tester.class;

Object tester=classType.newInstance();

Method addMethod=classType.getMethod("test",new Class[]{int.class,Integer.class});

Object result=addMethod.invoke(tester,

new Object[]{new Integer(100),new Integer(200)});

D. Class classType=Tester.class;

Object tester=classType.newInstance();

Method addMethod=classType.getMethod("test",new Class[]{int.class,Integer.class});

Integer result=addMethod.invoke(tester,

new Object[]{new Integer(100),new Integer(200)});

3、下列关于Java代码描述中正确的是（B）

A. 反射与内省并没有太大的关系

B. 反射是一个确定有哪些成员变量、成员方法可在一个对象上使用的过程

C. 内省（Introspector）是Java语言对JavaEE类属性、事件的处理方法

D. 内省对于任意一个对象，都能够调用它的任意一个方法

4、下列关于内省与反射描述错误的是（B）

A. 内省是一种特殊的反射

B. 反射技术操作javabean更简单，方便

C. 内省是基于反射的，只是在反射的基础之上封装了

D. 动态获取类信息以及动态调用类对象方法的功能叫做Java语言的反射机制。

5、java中内省与反射描述正确的是（D）

A．内省比反射机制更简单，更强大

B. 反射就是把java类中的各种成分映射成一个个的class代码类

C. 内省比反射的作用范围更广

D. 反射就是通过把指定的类中各种元素成分都映射成相关的反射包中的相应类

6、Java反射的功能? （C）

A. 通过反射不可以判断对象所在的类。  
　　B. 通过反射不能判断运行时对象所具有的成员变量和方法。

C.通过反射可以调用到private的方法

D. 通过反射不可以生成动态代理

7、内省的作用是：(A)

A.内省基于反射实现B.主要用于操作接口C.通过内省不可以获取bean的getter/setter

8、Java反射的类一般在哪个包里（B）

### A. [java.util.List](https://www.cnblogs.com/cuglkb/p/7027907.html)B. java.lang.relfect C. java.nio.channels D. java.util.concurrent

9、1. 关于类的加载时机说法错误的是(A )

A、Java程序启动时加载所有的类到内存中

B、某个类的对象被创建时加载一个类

C、用类名调用该类的静态方法时加载一个类

D、初始化一个类的子类时加载一个类

10. 关于反射机制下列说法错误的是(B )

A、反射机制指的是在程序运行过程中，通过.class文件加载并使用一个类的过程

B、反射机制指的是在程序编译期间，通过.class文件加载并使用一个类的过程

C、反射可以获取类中所有的属性和方法

D、暴力反射可以获取类中私有的属性和方法

11. 使用反射机制获取一个类的属性，下列关于getField()方法说法正确的是(A )

A、该方法需要一个String类型的参数来指定要获取的属性名

B、该方法只能获取私有属性

C、该方法只能获取公有属性

D、该方法可以获取私有属性，但使用前必须先调用setAccessible(true)

12.有关于反射说法错误的是(D)

A.使用Class.forName("com.yy.xxx")方法获取类

B.class.newInstance()实例化一个对象

C.class.getDeclaredMethods()获取方法列表

D.method.invoke(obj, args)不能执行私有方法

13、已知对象obj的类型是字符串， 获取obj的Class对象的方法包括（ BCD ）

A. obj.class(); B. obj.getClass(); C. String.class ;

D. Class.forName("java.lang.String");

14、关于反射机制下列说法错误的是( B )  
A、反射机制指的是在程序运行过程中，通过.class文件加载并使用一个类的过程  
B、反射机制指的是在程序编译期间，通过.class文件加载并使用一个类的过程  
C、反射可以获取类中所有的属性和方法  
D、暴力反射可以获取类中私有的属性和方法

15、Java提供了一套API来访问某个属性的getter/setter方法，这些API存放在（ C ）中。

A、java.lang B、java.util C、java.beans D、java.string

16、下列哪个方法能够获取一个类中私有的方法（D）

A. getFields()

B. getDeclaredFields()

C. getMethod()

D. getDeclareMethod()

1. 多选题
2. 以下哪些方法在Class类中定义？（A、C）
3. getConstructors() B. getPrivateMethods() C. getDeclaredFields()
4. getImports();
5. 以下哪些说法正确？（B、C、D）
6. 动态代理类与静态代理类一样，必须由开发人员编写源代码，并编译成.class文件
7. JDK动态代理类与被代理类具有同样的接口
8. java.lang.Exception类实现了java.io.Serializable接口，因此Exception对象可以被序列化后在网络上传输
9. java.lang.reflect包中的Proxy类提供了创建动态代理类的方法

3、 以下关于反射与内省机制错误的是？（A、C）

A. Java中一般由反射机制发挥作用

B. 通过使用内省，我们可以确定一个对象中哪些方法适用于被其他对象访问

C. 内省对于任意一个对象，都能够调用它的任意一个方法

D. Spring、Hibernate等等集合无一不使用了反射机制

4、以下哪些动态性质？（A、B、C）

A. 运行时生成对象实例;B.运行期间调用方法；C.运行时更改属性

D.运行时生成方法

5. Java反射机制的作用？（A、B、C、D）

A.在运行时判断任意一个对象所属的类。B.在运行时判断任意一个类所具有的成员变量和方法。

C.在运行时任意调用一个对象的方法 D.在运行时构造任意一个类的对象

6、Java内省类库有（A、B、C）

A. java.beans.Introspector B. java.beans.BeanInfo

C. java.beans.PropertyDescriptor D. java.util.concurrent

7、JAVA反射机制主要提供了以下哪些功能？（A B C D）

A.在运行时判断一个对象所属的类

B.在运行时构造一个类的对象

C.在运行时判断一个类所具有的成员变量和方法

D.在运行时调用一个对象的方法

8.以下哪些方法在Class类中定义？（A C）

A．getConstructors() B．getPrivateMethods() C．getDeclaredFields()

D．getImports() E．setField()

9、Java反射机制主要提供了以下哪些功能？（A、B、C、D）

A、在运行时判断一个对象所属的类

B、在运行时构造一个类的对象

C、在运行时判断一个类所具有的成员变量和方法

D、在运行时调用一个对象的方法

10、反射的功能（ ABCDE ）。

A、在运行时创建新类对象

B、在运行时调用任一个对象的方法

C、在运行时构造任意一个类的对象

D、在运行时判断任意一个对象所属的类

E、在运行时判段任意一个类所具有的成员变量和方法

11、下列哪些是获取Class类的对象的方法？（A、D、E）

A． Class.forName(全限定名) B. Clazz.newInstance()

C. clz.getFields() D. 对象.getClass() E. 类名.class

1. 填空题（每空1分，共10分）
2. 动态代理类都具有一个public 类型的构造方法，该构造方法有一个\_\_\_\_\_\_\_\_\_\_类型的参数。(InvocationHandler)
3. 反射在运行时可以获取到.class的任何定义信息，包括（\_\_\_\_\_\_\_\_，\_\_\_\_\_\_\_\_，\_\_\_\_\_\_\_\_）。

(成员变量, 成员方法,构造方法)

1. 在Java程序中许多对象在运行时都会出现两种类型：**编译时类型**和**运行时类型**
2. Java反射中使用Class对象的\_\_\_\_\_\_\_\_\_\_ 方法来创建Class对象对应类的实例。调用 \_\_\_\_\_\_\_\_\_\_方法。(newInstance()、invoke())
3. 反射的英文单词是 Reflection ；内省的英文单词是 Introspector 。
4. JAVA反射机制是在运行状态中，对于任意一个对象，都能够调用它的任意一个（方法）和（属性）
5. 获取Class对象的三种方式 （类名.class），对象名.getClass() ，Class.forName(全类名)
6. （类装载器）是用来把类(class)装载进 JVM 的
7. 获取字符串数组的Class对象： 。
8. Class cSArray = Class.forName("[Ljava.lang.String;");
9. 获取Double数组的Class对象： Class cDoubleArray = Class.forName("[D");
10. 获取字符串二维数组的Class对象： Class cS2Array = Class.forName("[[Ljava.lang.String;");
11. 获取Double的Class对象： Class c = Double.class;
12. 获取字符串的父类的Class对象： Class c = String.class.getSuperclass();
13. 如何通过java反射机制实现知道一个全限类名创建一个对象\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（Class.forName("全限类名")）
14. Class类中通过无参构造创建对象的方法为\_\_\_\_\_\_\_\_\_\_。(newInstance())
15. 判断题
16. JDK的动态代理类是public、final和非抽象类型的。（ T ）
17. Java内省机制容许程序在运行时加载、探知、使用编译期间完全未知的classes。（ N ）
18. JDK的动态代理类是public、final和非抽象类型的。（ T ）
19. 反射机制的优点就是可以实现动态创建对象和编译，体现出很大的灵活性（T）
20. 在Java反射功能中用instanceof关键字来判断是否为某个类的实例（ T ）。
21. 内省（Introspector）是[Java语言](https://www.baidu.com/s?wd=Java%E8%AF%AD%E8%A8%80&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YknAf4rAmdm10LuymvP1fY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjnYnW63n1ns" \t "_blank)对Bean类属性、事件的一种缺省处理方法。（ T ）。
22. 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（T）
23. JAVA反射机制是在运行状态中，对于任意一个对象，都能够调用它的任意一个方法和属性（T）
24. 反射能在运行时判断一个对象所属的类（T）
25. 反射能在运行时判断一个类所具有的成员变量和方法（T）
26. getDeclaredMethod()可以获取私有方法。 （ T ）
27. getDeclaredMethod()可以获取继承方法。 ( F ）
28. getMethod()可以获取继承方法。 （ F ）
29. getMethod()可以获取继承方法。 （ T ）
30. 构造方法可以继承。 （ F ）
31. 反射的优点是可以动态的创建对象和编译，最大限度发挥了java的灵活性。（ T ）
32. Class的方法getMethods()能够获得类的public类型的方法。（T）
33. Class的方法newInstance()表示通过类的任意构造方法创建这个类的一个对象。（F）
34. Java反射机制容许程序在运行时加载、探知、使用编译期间完全未知的classes。换言之，Java可以加载一个运行时才得知名称的class，获得其完整结构。（ √ ）
35. Class类的getMethod()方法能够获取类中声明的任何方法。（ F ）
36. 内省是通过反射来实现的，对JavaBean类属性、事件的一种缺省处理方法。（ T ）
37. 类中私有的属性或者方法，使用反射不可以直接调用。（ T ）

五、简答题

1、什么是反射机制？

答案：反射机制值得是程序在运行时能够获取自身的信息。在java中，只要给定类的名字，那么就可以通过反射机制来获得类的所有信息

2、Java反射机制提供了什么功能，它有什么优缺点？

答案：

Java反射机制提供如下功能：

在运行时能够判断任意一个对象所属的类

在运行时构造任意一个类的对象

在运行时判断任意一个类所具有的成员变量和方法

在运行时调用任一对象的方法

在运行时创建新类对象

反射机制的优点就是可以实现动态创建对象和编译，体现出很大的灵活性，特别是在J2EE的开发中它的灵活性就表现的十分明显。比如，一个大型的软件，不可能一次就把把它设计的很完美，当这个程序编译后，发布了，当发现需要更新某些功能时，我们不可能要用户把以前的卸载，再重新安装新的版本，假如这样的话，这个软件肯定是没有多少人用的。采用静态的话，需要把整个程序重新编译一次才可以实现功能的更新，而采用反射机制的话，它就可以不用卸载，只需要在运行时才动态的创建和编译，就可以实现该功能。

它的缺点是对性能有影响。使用反射基本上是一种解释操作，我们可以告诉JVM，我们希望做什么并且它满足我们的要求。这类操作总是慢于只直接执行相同的操作。

* 3、请简述关于反射的功能（5分）

答案：（1）在运行时判断任意一个对象所属的类 如 aclass.getClass() == bclass.getClass()

（2）在运行时构造任意一个类的对象 如Class.forName(className).netInstance()，className是运行时才得出来的类名

（3）在运行时判段任意一个类所具有的成员变量和方法

（4）在运行时调用任一个对象的方法

（5）在运行时创建新类对象 。

4、Class类的newInstance()方法的使用？

答案：根据调用构造方法的不同，用反射机制来实例化一个类，可以有两种途径。如果使用无参数的构造方法，则直接使用Class类的newInstance()方法即可。若需要使用特定的构造方法创建对象，则需要先获取Contructor实例，再用newIntance()方法创建对象。

5、可以在运行时获得对象的哪些类型信息？

答案：

，这个对象的所属的类（class）

这个类的构造器（constructor）

这个类中声明的方法（method）

这个类中声明的属性（field）

这个类中声明的注解（annotation）

这个类的父类（superclass）

这个类实现的 接口（interface）

6、简述java反射机制的作用。（5分）

答案： 1)在运行时判断任意一个对象所属的类

2)在运行时构造任意一个类的对象

3)在运行时判断任意一个类所具有的成员变量和方法

4)在运行时调用任意一个对象的方法

反射就是动态加载对象，并对对象进行剖析。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，这种动态获取信息以及动态调用对象方法的功能成为Java反射机制。

优点：可以动态的创建对象和编译，最大限度发挥了java的灵活性。

缺点：对性能有影响。使用反射基本上一种解释操作，告诉JVM我们要做什么并且满足我们的要求，这类操作总是慢于直接执行java代码。

7、内省和反射有什么区别？

反射：

反射就是运行时获取一个类的所有信息，可以获取到.class的任何定义的信息（包括成员 变量，成员方法，构造器等）

可以操纵类的字段、方法、构造器等部分。

内省：

内省基于反射实现，主要用于操作JavaBean，通过内省 可以获取bean的getter/setter

8、请问Java的反射，你目前主要用他做什么

答案：反射的用途 1. 各种开发框架中，如Spring。 2. 创建数据库链接时，如Class class=Class.forName(“com.java.dbtest.TestConnection”) 3.在IDE中对象.属性，这时候会用到反射

8、什么是Java的反射？它有什么作用？

答：

Java的反射是一套API， 通过这套API在Java程序运行时， 可以检测或修改方法， 类和接口的行为。

用于反射功能的的类放置在包java.lang.reflect下。

通过反射， 我们可以知道对象的类型，对象可以执行哪些方法等。

通过反射， 我们可以在运行时调用对象的方法。

9、 什么是Java的内省？ 它与Java的反射有什么联系？

答：

Java的内省是一个自动的处理过程， 在这个过程中会自动的分析一个bean的设计模式， 揭露这个bean的属性， 事件和方法。 这一个过程控制着bean操作和属性的发布和发现。

Java的内省使用反射机制， 它们两者之间的联系类似于JavaBeans 和其他 Java 类。

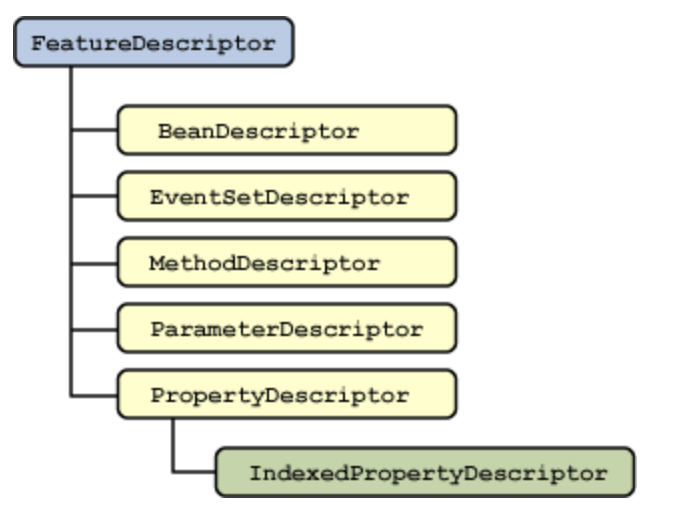
10、 谈谈Java内省的主要API

答：

包java.beans中的接口BeanInfo定义了一套方法，提供了关于bean的信息。

Introspector类的方法可以被构建工具和其他自动化环境使用， 以便获取bean的详细信息。

Introspector 类提供了描述类， 用于描述一个bean的属性， 事件， 和方法。



上图的每个类都描述了一个bean特定的属性。

# 11、简述内省机制和反射机制什么区别。（5分）

答案：内省操作只针对JavaBean，只有符合JavaBean规则的类的成员才可以采用内省API进行操作，而反射则不同，一个类的所有成员都可以进行反射操作。

内省是先得到属性描述器PropertyDecriptor后再进行各种操作，反射则是先得到类的字节码Class后再进行各种操作的。

12、什么是反射？

答：Java反射机制是在运行中，对任意一个类，能够获取得到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；这种动态获取类信息以及动态调用类对象方法的功能叫做Java语言的反射机制。

13、什么是内省？

答：内省是Java语言对JavaBean类属性、事件的处理方法。例如类User中有属性name，那么必定有getName，setName方法，我们可以通过他们来获取或者设置值，这是常规操作。

14、有何区别？

答：反射可以操作各种类的属性，而内省只是通过反射来操作JavaBean的属性；内省设置属性值肯定会调用seter方法，反射可以不用。

15、什么是反射？

答案：反射就是动态加载对象，并对对象进行剖析。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，这种动态获取信息以及动态调用对象方法的功能成为Java反射机制。

16、 反射的优缺点？

答案：优点：反射提高了程序的灵活性和扩展性,在底层框架中用的比较多，业务层面的开发过程中尽量少用。

缺点：对性能有影响。使用反射基本上一种解释操作，告诉JVM我们要做什么并且满足我们的要求，这类操作总是慢于直接执行java代码。

17、 Java反射机制的作用？

答案：1.在运行时判断任意一个对象所属的类

2.在运行时构造任意一个类的对象

3.在运行时判断任意一个类所具有的成员变量和方法

4.在运行时调用任意一个对象的方法

六、程序实现题

1、要求编写一个反射的抽象类。在这个类中实现两个基本的方法，通过子类属性名赋值和取值。（属性名的定义符合POJO定义）

|  |
| --- |
| **class** ReflactBase {  @SuppressWarnings("unchecked")  **public** **void** putValue(String name, Object value) {  Class c = **this**.getClass();  Class v = value.getClass();  **try** {  Method m = c.getMethod("set" + name, **new** Class[] { v });  m.invoke(**this**, **new** Object[] { value });  } **catch** (Exception e) {  e.printStackTrace();  }  }  @SuppressWarnings("unchecked")  **public** Object outValue(String name) {  Class c = **this**.getClass();  Object o = **null**;  **try** {  Method m = c.getMethod("get" + name, **new** Class[] {});  o = m.invoke(**this**, **new** Object[] {});  } **catch** (Exception e) {  e.printStackTrace();  }  **return** o;  }  }  **class** TestObj **extends** ReflactBase{  **private** String name;  **private** Integer age;  **private** Date birthday;  **private** String address;  **private** String email;  **private** String phone;  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** Integer getAge() {  **return** age;  }  **public** **void** setAge(Integer age) {  **this**.age = age;  }  **public** Date getBirthday() {  **return** birthday;  }  **public** **void** setBirthday(Date birthday) {  **this**.birthday = birthday;  }  **public** String getAddress() {  **return** address;  }  **public** **void** setAddress(String address) {  **this**.address = address;  }  **public** String getEmail() {  **return** email;  }  **public** **void** setEmail(String email) {  **this**.email = email;  }  **public** String getPhone() {  **return** phone;  }  **public** **void** setPhone(String phone) {  **this**.phone = phone;  }  @Override  **public** String toString() {  **return** name+","+age+","+email+","+phone+","+address;  }  }  **public** **class** RefleTest {  @SuppressWarnings("unchecked")  **public** **static** **void** main(String[] args) **throws** Exception {  //模拟获取的excel数据  List<String> list = **new** ArrayList<String>();  List<HashMap<String,Object>> list2 = **new** ArrayList();  list.add("Name");  list.add("Age");  list.add("Birthday");  list.add("Email");  list.add("Phone");  list.add("Address");  HashMap<String,Object> map = **new** HashMap<String,Object>();  map.put("Name", "徐长今");  list2.add(map);  map = **new** HashMap<String,Object>();  map.put("Age", 16);  list2.add(map);  map = **new** HashMap<String,Object>();  map.put("Birthday", **new** Date());  list2.add(map);  map = **new** HashMap<String,Object>();  map.put("Email", "changjin@hanguo.com");  list2.add(map);  map = **new** HashMap<String,Object>();  map.put("Phone", "13312312301");  list2.add(map);  map = **new** HashMap<String,Object>();  map.put("Address", "首尔");  list2.add(map);        //第一种方式  TestObj o = **new** TestObj();  o.putValue("Name", "林晚荣");  o.putValue("Email", "abc@125.com");  o.putValue("Phone", "120807756");  o.putValue("Address", "地址");  o.putValue("Age", 24);  o.putValue("Birthday", **new** Date());  System.*out*.println(o.toString());    //第二种方式  o=**new** TestObj();  **for**(**int** i=0;i<list2.size();i++){  HashMap<String,Object> hm = list2.get(i);  //存入类中  o.putValue(list.get(i), hm.get(list.get(i)));  }  System.*out*.println(o.toString());        //获取值  **for**(**int** i=0;i<list.size();i++){  System.*out*.print(list.get(i)+":"+o.outValue(list.get(i))+" ");  }  }  } |

2、反射获取： 通过反射获取类，实现三种不同方法（10分）

|  |
| --- |
| public void test() throws Exception {  // 通过class.forName加载类  Class cla = Class.forName("com.cn.reflect.Person");  // 直接通过类名.class  Class cla1 = Person.class;  // 通过对象名.getClass()方法  Class cla2 = new Person().getClass();  } |

3、实现一个方法：

|  |
| --- |
| public static Object execute(String className, String methodName, Object args[]) |

实现 “通过类的名字、方法名字、方法参数调调用方法，返回值为该方法的返回值。” 的功能。

public static Object execute(String className, String methodName, Object args[]) {

Object results = null;

try {

// 获得类的Class对象

Class<?> clazz = Class.forName(className);

Method method = null;

for (int i = 0; i < clazz.getMethods().length; i++) {

// 返回该类的所有方法

method = clazz.getMethods()[i];

// 查找名字匹配的方法

if(methodName.equals(method.getName())) {

// 创建临时对象，调用方法

results = method.invoke(clazz.newInstance(), args);

break;

}

}

} catch (Exception e) {

e.printStackTrace();

}

return results;

}

public static void main(String[] args){

Object result=MethodInvoker.execute("Account", "save", new Object[]{new Double(1000)});

System.out.println("存款1000后，余额为： "+result);

}

}

4、编写一段Java反射代码（**无标准答案，能体现反射原理即可**）

（10分）

|  |
| --- |
| public static void main(String[] args) {  try {  Class c=Class.forName("java.util.HashSet");  Object o=c.newInstance();  Method[] methods=c.getDeclaredMethods();  for(Method method:methods){  System.out.println(method);  }  Method m1=c.getMethod("add", Object.class);  m1.invoke(o, "cyq");  m1.invoke(o, "hello");  m1.invoke(o, "java");  System.out.println(o);  } catch (Exception e) {  e.printStackTrace();  }  } |

5、建立一个类Test， 包含一个私有的成员字符串变量s， 一个公共无參构造方法，

一个公共的无參方法method1, 一个公共的有參的方法method2(int n), 一个私有的无參方法method3()。 然后通过反射机制， 建立一个Test对象， 并调用这个对象的所有方法， 设置对象中s的值。 （10分）

|  |
| --- |
| class Test  {      private String s;     public Test()  {  s = "GeeksforGeeks"; }    public void method1()  {          System.out.println("The string is " + s);      }      public void method2(int n)  {          System.out.println("The number is " + n);      }        private void method3() {          System.out.println("Private method invoked");      }  }  public static void main(String args[]) throws Exception      {            Class cls = Test.class;          System.out.println("The name of class is " +                              cls.getName());    Test obj = (Test)cls.newInstance();          Constructor constructor = cls.getConstructor();          System.out.println("The name of constructor is " +                              constructor.getName());            System.out.println("The public methods of class are : ");            Method[] methods = cls.getMethods();            // 打印对象中方法的名称          for (Method method:methods)              System.out.println(method.getName());              // 获得method2          Method methodcall2 = cls.getDeclaredMethod("method2",                                                   int.class);            // 调用method2          methodcall2.invoke(obj, 19);              // 获得成员变量s          Field field = cls.getDeclaredField("s");            // 设置可以访问私有成员变量          field.setAccessible(true);            // 设置对象中s的值          field.set(obj, "JAVA");            // 获得method1          Method methodcall1 = cls.getDeclaredMethod("method1");            // 在运行时调用method1          methodcall2.invoke(obj);            // 获得方法method3          Method methodcall3 = cls.getDeclaredMethod("method3");            // 设置可以访问私有方法method3          methodcall3.setAccessible(true);            // 在运行时调用method3          methodcall3.invoke(obj);      } |

* 1. 定义一个简单的类， 演示Java的内省

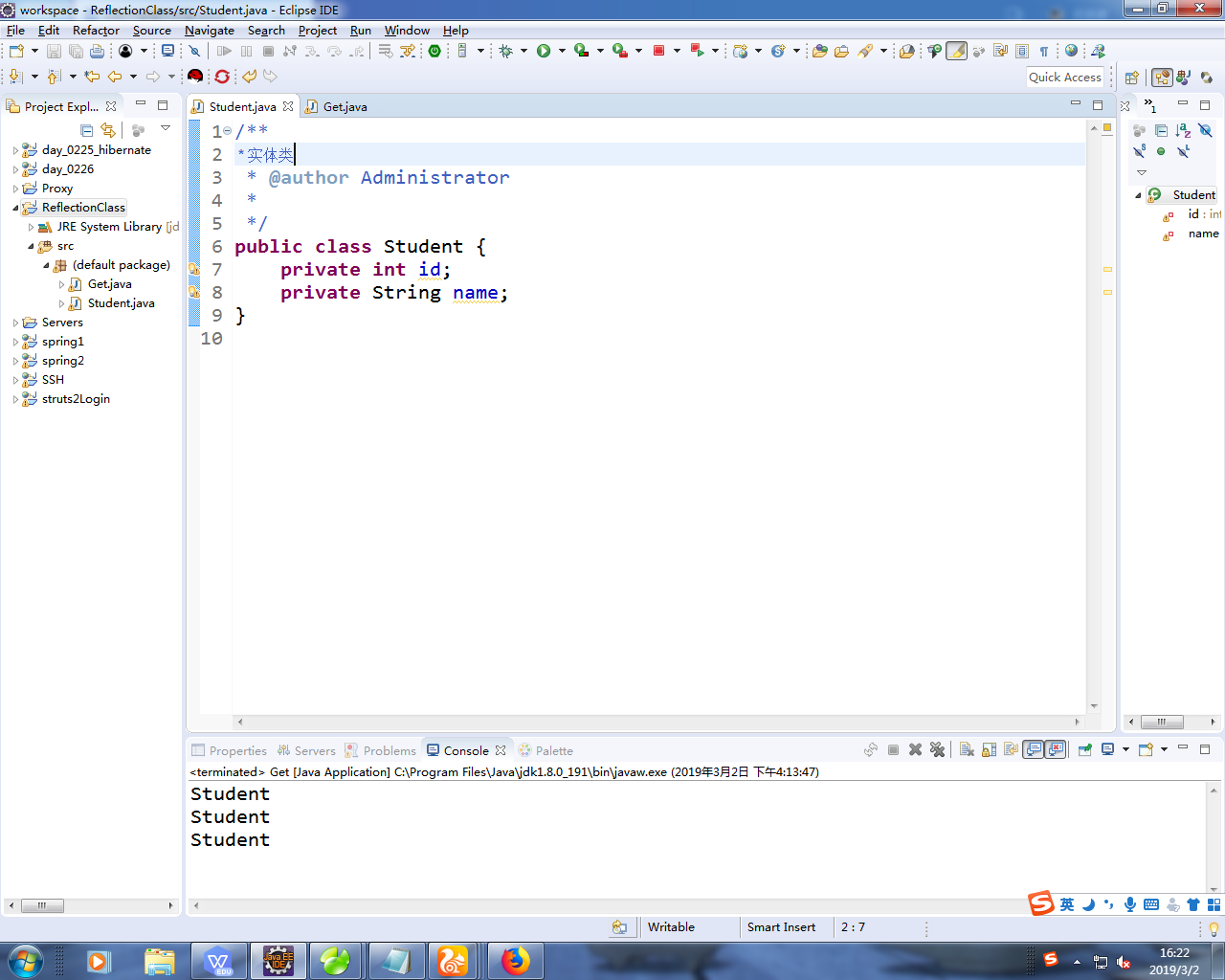
|  |
| --- |
| public class SimpleBean  {  private final String name = "SimpleBean";  private int size;  public String getName()  {  return this.name;  }  public int getSize()  {  return this.size;  }  public void setSize( int size )  {  this.size = size;  }  public static void main( String[] args )  throws IntrospectionException  {  BeanInfo info = Introspector.getBeanInfo( SimpleBean.class );  for ( PropertyDescriptor pd : info.getPropertyDescriptors() )  System.out.println( pd.getName() );  }  }  输出：  class  name  size  其中class属性是继承Object类。 |

7、写一个方法，此方法可将obj对象中名为propertyName的属性的值设置为value

（10分）

|  |
| --- |
| public static void setProperty(Object obj, String propertyName, Object value) throws Exception {  Class c = obj.getClass();  Field f = c.getDeclaredField(propertyName);  // 获得包括私有和公有的属性  f.setAccessible(true);  // 还要配合这个，取消权限检查，才能设置属性  f.set(obj, value);  System.out.println(obj);  } |

8、创建一个实体类，一个测试类，通过java反射机制三种方式获取实体类的类名



9、写一个Student类，包含属性name和age，无参构造方法。使用反射的方式，获取构造方法，创建对象，为name和age赋值，最后输出对象信息

（10分）

|  |
| --- |
| **public** **class** Student {  **private** String name;  **private** **int** age;    **public** Student() {  }    **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** "Student [name=" + name + ", age=" + age + "]";  }    }  **public** **class** StudentTest {  **public** **static** **void** main(String[] args) **throws** Exception, SecurityException {  //获取Class对象  Class<Student> clazz = Student.**class**;  //获取无参构造  Constructor<Student> constructor = clazz.getConstructor(**null**);  //通过构造方法创建对象  Student s = (Student) constructor.newInstance();  //获取setName方法  Method setNameMethod = clazz.getMethod("setName", String.**class**);  //调用setName方法  setNameMethod.invoke(s, "张三");  //获取setAge方法  Method setAgeMethod = clazz.getMethod("setAge", **int**.**class**);  //调用setAge方法  setAgeMethod.invoke(s, 20);  //输出对象信息  System.***out***.println(s);  }  } |