目标:

* Java反射机制概念和实现（重点理解）
* 利用反射机制实现方法的动态调用

一、理解Java的反射机制

1.什么是反射机制:

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；

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。

2.Java反射机制主要提供了以下功能：

在运行时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；

在运行时调用任意一个对象的方法；生成动态代理。

3．理解反射API(位于java.lang.reflect包中)

Class类 类|字节码

Method类 方法

Field类 属性

1. 实现java的反射机制

|  |
| --- |
| **public static void** main(String[] args) {  *//创建一个学生对象存储学生信息并显示  //1.创建学生对象  //Student stu=new Student();  //赋值  //stu.setXh(101);  //stu.setName("张三");  //stu.setAge(21);  //显示  //stu.show();* **try** {  *//1.实现反射(理解为获取类的字节码class)  //Class c=Class.forName("类的限定名");  //或者  //Class c=Student.class;* Class c=Class.*forName*(**"com.kgc.ch13.Student"**);   *//2.利用反射实现以下功能  //2.1 动态创建类的对象* Object o=c.newInstance();   *//2.2 动态获取类的所有属性* Field [] fields=c.getDeclaredFields();  **for** (Field field:fields) {  *//System.out.println("属性类型:"+field.getType());* System.***out***.println(**"属性名称:"**+field.getName());  }  *//2.3 动态给属性赋值  //Field nameFiled=c.getDeclaredField("name");  //nameFiled.set(o,"张三"); //赋值   //2.4 动态获取所有类的方法  //Method []methods=c.getMethods(); //获取当前类中所有方法（包括继承的方法）* Method []methods=c.getDeclaredMethods(); *//所有当前类中定义的所有方法* **for** (Method method:methods) {  System.***out***.println(**"方法名称:"**+method.getName());  }   *//2.5 动态获取某个方法进行调用* Method methodXh=c.getDeclaredMethod(**"setXh"**,**int**.**class**);  Method methodName=c.getDeclaredMethod(**"setName"**,String.**class**);  Method methodAge=c.getDeclaredMethod(**"setAge"**,**int**.**class**);  Method methodShow=c.getDeclaredMethod(**"show"**);  *//调用方法给私有属性赋值* methodXh.invoke(o,101);  methodName.invoke(o,**"张三"**);  methodAge.invoke(o,21);  *//显示* methodShow.invoke(o);   } **catch** (ClassNotFoundException e) {  e.printStackTrace();  } **catch** (IllegalAccessException e) {  e.printStackTrace();  } **catch** (InstantiationException e) {  e.printStackTrace();  } **catch** (NoSuchMethodException e) {  e.printStackTrace();  } **catch** (InvocationTargetException e) {  e.printStackTrace();  } } |

1. 反射的应用:动态实现方法的调用(理解)

动态方法调用:通过方法名称在运行时决定调用的方法

|  |
| --- |
| **public class** Dog {  *//跑* **public void** run(){  System.***out***.println(**"狗正在跑....."**);  }  *//睡觉* **public void** sleep(){  System.***out***.println(**"狗正在睡觉....."**);  }  *//摇尾巴* **public void** yao(){  System.***out***.println(**"狗看见主人回家啦!不停的摇尾巴....."**);  } }  **public class** TestDog {  **public static void** main(String[] args) {  *//创建对象  /\* Dog dog=new Dog();  dog.run(); //重新编译，在运行 硬编码\*/   //运行决定调用哪个方法==>称为方法的动态调用* Scanner input=**new** Scanner(System.***in***);  **while**(**true**){  System.***out***.println(**"请输入方法名称:"**);  String methodName=input.nextLine(); *//run  //Dog dog=new Dog();  //dog.methodName();* **try** {  *//利用反射实现方法动态调用方法* Class c=Class.*forName*(**"com.kgc.ch13.Dog"**);  Object o=c.newInstance();  *//动态获取类的方法进行调用* Method method=c.getDeclaredMethod(methodName);  method.invoke(o);  } **catch** (ClassNotFoundException e) {  e.printStackTrace();  } **catch** (IllegalAccessException e) {  e.printStackTrace();  } **catch** (InstantiationException e) {  e.printStackTrace();  } **catch** (NoSuchMethodException e) {  e.printStackTrace();  } **catch** (InvocationTargetException e) {  e.printStackTrace();  }  }  } } |

1. 扩展：更新xml文档中的内容

|  |
| --- |
| **public static void** main(String[] args) **throws** Exception {  *//更新xml文档===添加标签  //1.读取xml文件加载到内存* File file=**new** File(**"E:/IdeaProject/One/hjava/src/stu.xml"**);  SAXReader rader=**new** SAXReader();  Document doc=rader.read(file);   *//2.添加节点(标签)  //创建元素* Element root=doc.getRootElement();  Element student=root.addElement(**"student"**);  Element name=student.addElement(**"name"**); *//添加元素* name.setText(**"张三1"**); *//设置广本* Element age=student.addElement(**"age"**);  age.setText(**"22"**);  Element sex=student.addElement(**"sex"**);  sex.setText(**"男"**);  student.addAttribute(**"xh"**,**"91"**); *//添加属性   //3.保存文件* FileWriter fw=**new** FileWriter(file);  OutputFormat format=OutputFormat.*createPrettyPrint*(); *//指定标签的格式* XMLWriter xw=**new** XMLWriter(fw,format);  xw.write(doc); *//保存* xw.close();  fw.close();  System.***out***.println(**"保存成功"**); } |
| **public static void** main(String[] args) **throws** Exception {  *//更新xml文档===删除标签  //1.读取xml文件加载到内存* File file=**new** File(**"E:/IdeaProject/One/hjava/src/stu.xml"**);  SAXReader rader=**new** SAXReader();  Document doc=rader.read(file);  Element root=doc.getRootElement();   *//2.删除节点(标签) 通过学号删除学生* List<Element> childs=root.elements();  **for** (Element element:childs) {  *//通过子标签的名称获取标签内容* String xh = element.attributeValue(**"xh"**);  **if**(xh.equals(**"105"**)){  *//删除标签 父标签移除子标签* element.getParent().remove(element);  }  }   *//3.保存文件* FileWriter fw=**new** FileWriter(file);  OutputFormat format=OutputFormat.*createPrettyPrint*(); *//指定标签的格式* XMLWriter xw=**new** XMLWriter(fw,format);  xw.write(doc); *//保存* xw.close();  fw.close();  System.***out***.println(**"保存成功"**);  } } |