1. 反射
2. 概述

将类的各个组成部分封装为对象，这是反射机制。类是由成员变量，成员函数，构造函数组成的，反射将类的这些组成部分都封装成对象。

1. 具体流程
2. （源代码阶段）源码经过javac编译后生成了.class字节码文件，此时还在硬盘中，要创建该类对象，就要在内存中创建类对象，则就需要类加载器将字节码文件加载到内存中。
3. （Class类对象阶段）内存中用Class类对象来描述加载到内存的字节码文件，将成员变量封装为Field对象，将构造方法封装为Constructor对象，将成员方法封装为Method对象。
4. （运行阶段）加载进内存的Class类对象后，就可以创建类对象了。
5. 优点
6. 可以在程序运行过程中，操作这些对象
7. 解耦合，提高程序的可扩展性
8. 获取Class对象的三种方式

同一个字节码文件（.class）在一次运行过程中只会被加载一次，因此不论通过哪种方式获取的Class对象只有一个。

1. Class.for(“全类名”)

当字节码文件还未加载到内存中时，通过Class.for(“全类名”)可以将字节码文件加载到内存中，并返回Class对象。这种方式多用于配置文件，将类名定义在配置文件中，读取文件，加载类。

所谓的全类名指的是从包的名字开始一直到类的名字

如：

Class<?> cls = Class.*forName*("反射.Person");  
System.*out*.println(cls);

1. 类名.class

当类已经存在时，通过类名的属性class可以获取到该类的Class对象。这种方式多用于参数的传递。

如：

Class cls2 = Person.class;  
System.*out*.println(cls2);

1. 对象.getClass()

当类对象已经存在时，可以通过类对象的getClass()方法获取Class对象。这种方式多用于对象的获取字节码。

如：

Person p = new Person();  
Class cls3 = p.getClass();  
System.*out*.println(cls3);

1. Class对象功能
2. 获取功能

1. 获取成员变量们

\* Field[] getFields() ：获取所有public修饰的成员变量

\* Field getField(String name) 获取指定名称的 public修饰的成员变量

\* Field[] getDeclaredFields() 获取所有的成员变量，不考虑修饰符

\* Field getDeclaredField(String name)

示例：

package 反射;  
  
import java.lang.reflect.Field;  
  
public class ReflectDemo {  
 public static void main(String[] args) throws ClassNotFoundException, NoSuchFieldException, IllegalAccessException {  
 //1. 全类名  
 Class<?> cls = Class.*forName*("反射.Person");  
 System.*out*.println(cls);  
  
 //2.类名.class  
 Class cls2 = Person.class;  
 System.*out*.println(cls2);  
  
 //3.对象.getClass()  
 Person p = new Person();  
 Class cls3 = p.getClass();  
 System.*out*.println(cls3);  
  
 Field[] fields = cls.getFields();  
 for(Field field : fields){  
 System.*out*.println(field);  
 }  
// Field a = cls.getField("age");  
// Object val = a.get(p);  
// System.out.println(val);  
 Field[] deFields = cls.getDeclaredFields();  
 for(Field deField : deFields){  
 System.*out*.println(deField);  
 }  
  
 }  
}

Field成员变量的方法：

（1）设置值

Void set(Object obj, Object value)

（2）获取值

Get(Object obj)

（3）忽略访问权限修饰符的安全检查

setAccessible(true)，暴力反射，可以忽略权限的修饰符，访问private变量。

2. 获取构造方法们

\* Constructor<?>[] getConstructors() 返回所有公共构造方法对象的数组

\* Constructor<T> getConstructor(类<?>... parameterTypes) 注意这里的参数是对应参数类型的Class对象，返回单个公共构造方法对象

\* Constructor<T> getDeclaredConstructor(类<?>... parameterTypes) 返回单个构造方法对象

\* Constructor<?>[] getDeclaredConstructors() 返回所有构造方法对象的数组

Constructor<?> c1 = cls.getConstructor(String.class, int.class);  
System.*out*.println(c1);  
  
Object p1 = c1.newInstance("xx", 22);  
System.*out*.println(p1);

Constructor的方法：

创建对象：  
T newInstance(Object…initargs)

如果使用空参构造方法创建对象，可简化为Class对象的newInstatnce()。

3. 获取成员方法们：

\* Method[] getMethods() 返回所有公共成员方法对象的数组，包括继承的

\* Method getMethod(String name, 类<?>... parameterTypes) 返回单个公共成员方法对象

\* Method[] getDeclaredMethods() 返回所有成员方法对象的数组，不包括继承的

\* Method getDeclaredMethod(String name, 类<?>... parameterTypes) 返回单个成员方法对象

Method m1 = cls.getMethod("eat");  
m1.invoke(p);

Method的方法：

1. 执行方法

Object invoke(Object obj, Object… args)

Obj:调用方法的对象

Args:方法需要的参数

调用obj对象的成员方法，参数是args，

1. 获取方法名称

String getName()

4. 获取全类名

\* String getName()

1. 反射的实例

（一）需求

写一个“框架”，可以帮我们创建任意类的对象，执行其中的任意方法

1. 思路

框架的前提是不能改变该类的任何代码，但是依然可以创建任意类的对象，执行任意的方法，这里我们要通过修改另一个文件的方式来间接地修改代码，执行代码。这个文件就是配置文件，将需要创建的对象和方法定义在配置文件中，在程序中加载配置文件。

1. 步骤
2. 将需要创建的对象的全类名和需要执行的方法定义在配置文件中
3. 在程序汇总加载读取配置文件
4. 使用反射技术加载类文件进内存
5. 创建对象，执行方法
6. 代码实现
7. 创建一个Student类

类中只简单定义一个eat()方法

1. 创建pro,properties配置文件

配置文件中设置全类名和方法名。

className=反射.Student  
methodName=eat

1. 创建框架类

该类中按照步骤一步步执行，读取配置文件时通过类加载器来查找配置文件并读取。

1. 代码

package 反射;  
  
import java.io.IOException;  
import java.io.InputStream;  
import java.lang.reflect.InvocationTargetException;  
import java.lang.reflect.Method;  
import java.util.Properties;  
  
public class ReTest {  
 public static void main(String[] args) throws IOException, ClassNotFoundException, IllegalAccessException, InstantiationException, NoSuchMethodException, InvocationTargetException {  
  
 //加载配置文件  
 //创建Properties对象  
 Properties pro = new Properties();  
 //加载配置文件，转换为一个集合  
 //获取class目录下的配置文件  
 //获取该类的类加载器，并找到配置文件的字节流  
 ClassLoader classLoader = ReTest.class.getClassLoader();  
 InputStream is = classLoader.getResourceAsStream("pro.properties");  
 pro.load(is);  
  
 //获取配置文件中的数据  
 String className = pro.getProperty("className");  
 String methodName = pro.getProperty("methodName");  
  
 //加载该类进内存  
 Class cls = Class.*forName*(className);  
 //创建对象  
 Object obj =cls.getConstructor().newInstance();  
 //获取方法对象  
 Method method = cls.getMethod(methodName);  
  
 //执行方法  
 method.invoke(obj);  
  
 }  
}