# 基本概念

“反”指的是利用对象找到对象的出处,在Object类里有一个方法:

getClass()

**public final Class<?> getClass()**

# Class类

Java.lang

这个类是所有反射操作的源头,反射都要从这个类开始进行,这个类

**A.有三种实例化方式:**

1. 调用Object类中的getClass()方法(使用的概率很低,有实例化产生的情况下)
2. 使用”类名.class”取得
3. 调用Class类的一个方法(重点)

public static Class<?> forName(String className) throws ClassNotFoundException

和第二种方法比较:

此时可以不使用import 语句导入一个明确的类,而类名称是采用字符串的方式描述的

**B.Class类的一些成员方法:**

**1)String getName();返回完整的类名称**

**2)String getSimpleName();返回不含包名的类名称**

**3)Class<? super T> getSuperClass();返回继承的父类的Class实例,不能得到祖的Class实例**

**4)** **Class<?>[] getInterfaces();返回实现的接口的Class实例数组**

范例:3种实例化方式

|  |
| --- |
| Class<?> cls1 = new Date().getClass(); //第一种方法  Class<?> cls2 = Date.class; //第二种方法  Class<?> cls3 = Class.forName("java.util.Date"); //第三种方法 |

# 反射实例化对象

当拿到一个类的时候,肯定要直接使用new进行实例化对象,但是如果有了

Class对象,那么就可以利用反射来实例化对象

实例化对象的方法:

public T newInstance() throws InstantiationException,IllegalAccessException

范例:

|  |
| --- |
| class test{  public static void main(String args[]) throws Exception{  Class<?> cls = Class.forName("com.lw.Book"); //Class的泛型就写?  Object obj = cls.newInstance(); //调用无参构造,用Object来接受  Book book = (Book)obj;  System.out.print(book);  }  }  class Book{  @Override  public String toString() {  return "这是一本书";  }  public Book(){  System.out.println("Book无参构造");  }  } |

范例:反射的用处

|  |
| --- |
| package com.company;  interface Fruit{  public void eat();  }  class Apple implements Fruit{  @Override  public void eat() {  System.out.print("吃苹果");  }  }  class Factory{  public static Fruit getInstance(String className){  if("apple".equals(className)){  return new Apple();  }  return null;  }  }  public class TestFactory{  public static void main(String args[]){  Fruit f = Factory.getInstance("apple");  f.eat();  }  } |

如果要增加Fruit接口的子类,那么就还要修改工厂类

|  |
| --- |
| class Orange implements Fruit{  @Override  public void eat() {  System.out.print("吃橘子");  }  } |

|  |
| --- |
| class Factory{ //被修改  public static Fruit getInstance(String className){  if("apple".equals(className)){  return new Apple();  }else if("orange".equals(className)){  return new Orange();  }  return null;  }  } |

没增加一个类就要去修改工厂类,那么如果随时都可能增加子类呢？工厂类要被一直

进行修改,因为工厂类返回的对象都是通过new直接实例化的,要想解决这一问题,可以用反射解决

|  |
| --- |
| class Factory{  public static Fruit getInstance(String className){  Fruit f = null;  try {  f = (Fruit) Class.forName(className).newInstance();  }catch (Exception e){  e.printStackTrace();  }  return f;  }  }  Main函数里:  Fruit f = Factory.getInstance("com.company.Apple");  f.eat(); |

全部代码:

|  |
| --- |
| package com.company;  interface Fruit{  public void eat();  }  class Apple implements Fruit{  @Override  public void eat() {  System.out.print("吃苹果");  }  }  class Orange implements Fruit{  @Override  public void eat() {  System.out.print("吃橘子");  }  }  class Factory{  public static Fruit getInstance(String className){  Fruit f = null;  try {  f = (Fruit) Class.forName(className).newInstance();  }catch (Exception e){  e.printStackTrace();  }  return f;  }  }  public class TestFactory{  public static void main(String args[]){  Fruit f = Factory.getInstance("com.company.Apple");  f.eat();  }  } |

# 利用反射调用构造函数

取得构造函数:

取得全部构造: public Constructor<?>[] getConstructors() throws SecurityException

返回一个指定参数的构造: public Constructor<T> getConstructor(Class<?>... parameterTypes) throws NoSuchMethodException, SecurityException

**Constructor类**

**定义:**

java.lang.Object

java.lang.reflect.AccessibleObject

java.lang.reflect.Executable

java.lang.reflect.Constructor<T>

这个类提供了一个明确传递有参构造内容的实例化对象方法:

public T newInstance(Object... initargs)

throws InstantiationException,

IllegalAccessException,

IllegalArgumentException,

InvocationTargetException

范例:

|  |
| --- |
| package com.company;  public class Book{  private String title;  private double price;  public Book(String title,double price){  this.title = title;  this.price = price;  }  @Override  public String toString() {  return "书名:" + this.title + ",价格:" + this.price;  }  public Book(){  System.out.println("Book无参构造");  }  } |

|  |
| --- |
| package com.lw;  import java.lang.reflect.Constructor;  class test{  public static void main(String args[]) throws Exception{  //泛型都写?就行了  Class<?> cls = Class.forName("com.company.Book");  Constructor<?> con = cls.getConstructor(String.class,double.class);  Object obj = con.newInstance("java开发",34.6); //实例化对象  System.out.print(obj);  }  } |

# 利用反射调用普通方法

前提必须实例化一个对象:实例化对象的方式有三种:new,clone,反射

在Class类里面提供了取得普通方法的函数:

1)取得类中的全部方法: public Method[] getMethods()

throws SecurityException

2)取得类中的普通方法:

a. public Method getMethod(String name, //第一个参数是方法名称,第二个参数是参数类型.class

Class<?>... parameterTypes) 获取的方法为public类型的:

throws NoSuchMethodException,SecurityException

b. public Method getDeclaredMethod(String name, Class<?>... parameterTypes); 获取方法,不管方法的防控属性,如果要获取私有的方法,则先要setAccessible(true);取消封装

**Method类**

java.lang.Object

java.lang.reflect.AccessibleObject

java.lang.reflect.Executable

java.lang.reflect.Method

调用方法: public Object invoke(Object obj,Object... args) //第一个参数是实例化对象

throws IllegalAccessException,

IllegalArgumentException,

InvocationTargetException

范例:调用普通方法

|  |
| --- |
| package com.company;  public class Book{  private String title;  public void setTitle(String title){  this.title = title;  }  public String getTitle(){  return this.title;  }  } |

|  |
| --- |
| class test{  public static void main(String args[]) throws Exception{  String fieldName = "title";  Class<?> cls = Class.forName("com.company.Book");  Object obj = cls.newInstance(); //想要调用普通方法必须实例化对象  Method setMet = cls.getMethod("set" + initcap(fieldName), String.class);  Method getMet = cls.getMethod("get" + initcap(fieldName));  setMet.invoke(obj,"java开发");  System.out.print(getMet.invoke(obj));  }  public static String initcap(String str){  return str.substring(0,1).toUpperCase() + str.substring(1);  }  } |

# 反射调用属性

前提:实例化对象

Class里提供了两个方法来取得属性

1. 取得全部属性：public Field[] getDeclaredFields() throws SecurityException
2. 取得指定属性：public Field getDeclaredField(String name)

throws NoSuchFieldException,SecurityException

1. 取得public的属性: public Field getFiled(String name) throws NoSuchFieldException, SecurityException

**Field类**

java.lang.Object

java.lang.reflect.**AccessibleObject**

java.lang.reflect.Field

取得属性内容: public Object get(Object obj)

throws IllegalArgumentException,IllegalAccessException

设置属性内容: public void set(Object obj,Object value)

throws IllegalArgumentException,IllegalAccessException

**AccessibleObject类**

有一个设置属性是否封装的方法: public void setAccessible(boolean flag)

throws SecurityException

flag=true表示封装取消,构造方法和普通方法也同样可以取消封装

范例:属性

|  |
| --- |
| package com.company;  public class Book{  private String title;  } |

|  |
| --- |
| class test{  public static void main(String args[]) throws Exception{  String fieldName = "title";  Class<?> cls = Class.forName("com.company.Book");  Object obj = cls.newInstance();  Field f = cls.getDeclaredField("title");  f.setAccessible(true);//取消封装  f.set(obj,"java开发");  System.out.print(f.get(obj));  }  } |

# 类路径

类路径:就是class文件所在的目录

Class类里有一个方法:

public InputStream getResourceAsStream(String name) //通过一个类路径,返回一个InputStream

通过此方法,使代码的移植性更强,免得出现路径错误问题

|  |
| --- |
| Class c = Hello.class;  /\*  类路径:class文件所在的目录  \*/  InputStream in = c.getResourceAsStream("/jdbc.properties");  //“/”表示类路径的根目录  Properties pro = new Properties();  try {  pro.load(in);  } catch (IOException e) {  e.printStackTrace();  }  String name = pro.getProperty("name");  System.out.print(name); |

# 8.反射泛型

具体请看Demo3

# 9.反射注解

使用反射技术获取注解的信息

Author.java:

|  |
| --- |
| package anotation;  import java.lang.annotation.ElementType;  import java.lang.annotation.Retention;  import java.lang.annotation.RetentionPolicy;  import java.lang.annotation.Target;  /\*  @Target的作用是定义注解的使用范围  \*/  @Target({ElementType.METHOD,ElementType.FIELD})  @Retention(RetentionPolicy.RUNTIME)  public @interface Author {  //声明属性  String name() default "lw";  String modifiTime() default "2017-1-19"; //default后面接默认值  String[] address();//带有数组类型的属性  //如果注解的属性名称为value,并且没有其他的属性,可以不用写value=xxx,直接写xxx就可以了  String[] value();  } |

A.java:

|  |
| --- |
| package util;  import anotation.Author;  import org.junit.Test;  import java.lang.reflect.Constructor;  import java.lang.reflect.Method;  public class A {  @Author(name="eric",address = {"广州天河","湖北潜江"}, value = {"你好"})  public void save(){  /\*\*  \* 反射注解:使用反射的代码来获取注解上面的数据  \*/  // Class c;  // Method m;  // Field f;  Constructor con;  //c.getAnnotation();获取类上面的注解  //m.getAnnotation();获取方法上面的注解  //f.getAnnotation();获取属性上面的注解  //con.getAnnotation();获取构造函数上面的注解  try {  //1)得到save方法对象  Method m = this.getClass().getMethod("save",null);  //2)得到方法上面的注解  Author author = m.getAnnotation(Author.class);  //3)获取注解里的属性(数据)  String name = author.name();  String s = author.modifiTime();  String[] value = author.value();  System.out.println(name);  System.out.println(s);  System.out.println(value);  } catch (NoSuchMethodException e) {  e.printStackTrace();  }  }  @Author(name="eric",address = {"广州天河","湖北潜江"}, value = {"你好"})  private int age;  @Test  public void test() throws Exception{  save();  }  } |