·新章节

* 1. Object类中的 toString 方法

概述：Java.lang.object类是Java语言中的根类，即所有类的父类。它中描述的所有方法子类都可以使用。在对象实例化的时候，最终找的父类就是Object。

/\*  
java.long.Object类  
类 Object 是类层次结构的根（ 最顶层）类。每个类都可以使用Object作为父类。  
所有对象包括数组都实现这个类的方法。  
 \*/

定义一个 Person 标准类：

public class Person {  
 private String name;  
 private int age;  
  
 public Person() {  
 }  
  
 public Person(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
 /\*  
 直接打印对象的地址值没有意义，需要重写Object类的toString方法。  
 打印对象的属性(name,age)  
 \*/  
  
 /\* @Override // 覆盖重写。  
 public String toString() {  
 // return super.toString(); // 默认继承Object中的toString方法  
 return "Person{name = "+name+" ,age = "+age+"}";  
 }\*/  
  
 @Override  
 public String toString() {  
 return "Person{" +  
 "name='" + name + '\'' +  
 ", age=" + age +  
 '}';  
 }

// 省略了 Get/Set 方法

主函数：

/\*  
java.long.Object类  
类 Object 是类层次结构的根（ 最顶层）类。每个类都可以使用Object作为父类。  
所有对象包括数组都实现这个类的方法。  
 \*/  
public class Demo01ToString {  
  
 public static void main(String[] args) {  
 /\*  
 Person类默认继承了Object类，所以可以使用object类中的toString方法。  
 String toString() 返回该对象的字符串表示。  
 \*/  
 Person p = new Person("张三",18);  
 String s= p.toString();  
 // 直接打印对象的名字(打印的为地址值)，其实就是调用对象的toString方法 P = P.toString();  
 // Person 类中覆盖重写了Object中的toString方法 （abc)  
 System.out.println(p); // demo01.Person@3f99bd52 / abc / Person{name='张三', age=18}  
 System.out.println(s); // demo01.Person@3f99bd52 / abc / Person{name='张三', age=18}  
  
 /\*  
 看一个类是否重写了 toString 方法，直接打印这个类对象的名字即可。  
 如果没有重写 toString 方法，那么打印的就是对象的地址值(默认);  
 如果重写 toString 方法，那么就按照重写的方式打印。  
 \*/  
 Random r= new Random();  
 System.out.println(r); // java.util.Random@1218025c 没有重写 toString 方法。  
  
 ArrayList<Integer> list = new ArrayList<>(); // 重写了 toString 方法  
 list.add(1);  
 list.add(2);  
 System.out.println(list); // [1, 2]  
 }  
}

* 1. Object类中的 equals

Person 为标准类；

主函数：

public class Demo01Equals {  
 public static void main(String[] args) {  
 /\*  
 Person 默认继承了Object类，所有可以使用object类的equals方法。  
 boolean equals(Object obj)指示其他某个对象是否与此对象”相等“。  
 \*/  
 // 创建第一个对象  
 Person p1 = new Person("迪丽热巴",18);  
 // 创建第二个对象  
 Person p2 = new Person("古力娜扎",18);  
 System.out.println("P1:"+p1); // P1:demo01.Person@5674cd4d  
 System.out.println("P2"+p2); // P2demo01.Person@63961c42  
 p1 = p2; // true  
 /\* public boolean equals(Object obj) {  
 return (this == obj);  
 }  
 参数：  
 Object obj：可以传递任意对象  
 方法体：  
 == ：比较运算符，返回值的就是一个布尔值 true，false  
 基本数据类型：比较的是值。  
 引用数据类型：比较的时两个对象的地址值。  
 this是谁？ 哪个对象调用的方法，方法中的this就是哪个对象；P1调用的equals方法，所以this就是P1；  
 obj是谁？ 传递过来的参数P2；  
 this==obj --> P1==P2  
 \*/  
 boolean b = p1.equals(p2);  
 System.out.println(b); // false  
 }  
}

* 1. equals 的重写

/\*  
 Object 类的equals方法默认比较的是两个对象的地址值，没有意义。  
 所以我们要重写equals方法，比较两个对象的属性值(name,age)  
 对象的属性值一样，返回：true; 否则：返回false  
 问题：  
 隐含着一个多态  
 Object obj= P2 = new Person("古力娜扎”,18);  
 多态弊端：无法使用子类特有的内容（属性，方法）  
 解决：可以使用向下转型（强转）Object类型转换为Person  
 \*/  
类函数：（省略了上边的name、age的全参、无参、Get/Set）

/\* @Override  
 public boolean equals(Object obj) {  
  
 \*//\*  
 增加一个判断，是 Person 类型在转化，防止类型转换异常ClassCastException  
 \*//\*  
 if (obj instanceof Person) {  
 // 使用向下转型（强转）Object类型转换为Person  
 Person p = (Person) obj;  
 // 比较两个对象的属性：一个是调用方法的this(P1)，一个是P(obj==p2)  
 boolean b = this.name.equals(p.name) && this.age == p.age;  
 return b;  
 }  
  
 // 不是 Person 类型直接返回false  
 return false;  
 }\*/  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Person person = (Person) o;  
 return age == person.age &&  
 Objects.equals(name, person.name);  
 }  
  
 @Override  
 public int hashCode() {  
  
 return Objects.hash(name, age);  
 }

主函数：

public class Demo01Equals {  
 public static void main(String[] args) {  
 Person p1 = new Person("迪丽热巴",18);  
 Person p2 = new Person("古力娜扎",18);  
 boolean b = p1.equals(p2);  
 System.out.println(b); // false  
 }  
}

* 1. Objects 类

在 JDK 7 添加了一个Objects 工具栏，它提供了一些方法来操作对象，它由一些静态的实用方法组成。这些方法是null.save(空指针安全的)或null.tolerant(容忍空指针）,用于计算对象的ashcod返回对象的字符串表示形式，比较两个对象。

在比较两个对象的时候，Object 的 equals 方法容易抛出空指针异常，而Object类中的equals方法就优化了这个问题。

方法如下：

public static boolean equals(object a,object b); 判断对象是否相等。

主函数：

public class DemoMain01 {  
 public static void main(String[] args) {  
 String str1= "abc";  
 String str2 ="abc";  
 String str3 = "aaa";  
  
 boolean b = str1.equals(str2);  
 System.out.println(b); // true  
 System.out.println(str2.equals(str3)); // false  
  
 String str4 = null;  
 System.out.println(str4.equals(str3)); //NullPointerException  
  
 /\*  
 objects 类的 equals 方法：对两个对象进行比较，防止空指针异常。  
 public static boolean equals(Object a, Object b) {  
 return (a == b) || (a != null && a.equals(b));  
 }  
 \*/  
 boolean b2 = Objects.equals(str3,str4);  
 System.out.println(b2); // false  
 }  
}

类函数：（idea中自带的Objects类）

public static boolean equals(Object a, Object b) {  
 return (a == b) || (a != null && a.equals(b));  
}

* 1. 总结

1. Object类
2. Object类的介绍

Object类是所有类的父类，一个类都会直接或者间接的继承该类

该类中给我们提供了一些非常常用的方法。

1. toString() 方法

作用：打印对象的信息（重写前：打印包名称.类名称@地址值

重写后：打印的是对象中属性值）

类函数：（定义名字和年龄的标准类 + 重写 toString() 方法）

主函数：

package demo01;  
  
public class TesttoString {  
 public static void main(String[] args) {  
 String str = "abc"; // 在 String 中已经对toString方法进行了重写  
 /\*  
 String toString(){  
 return this;  
 }  
 this 就代表当前的一个字符串对象 所以 str == str.toString  
 \*/  
 System.out.println(str); // abc  
 System.out.println(str.toString()); // abc 默认调用 toString 方法。  
  
 Student stu = new Student("张三",12);  
 // 没有重写 toString 方法，默认继承 Object 中的toString方法（CTRL + Object）查找。  
 /\*  
 public String toString() {  
 return getClass().getName() + "@" + Integer.toHexString(hashCode());  
 }  
 包名称.类名称@哈希值  
 \*/  
 System.out.println(stu); // demo01.Student@3f99bd52 (地址值)  
  
 // 重写 toString 方法，使用快捷键 Alt + Ins  
 /\* @Override  
 public String toString() {  
 return "Student{" +  
 "name='" + name + '\'' +  
 ", age=" + age +  
 '}';  
 }\*/  
 System.out.println(stu); // Student{name='张三', age=12}  
 }  
}

1. equals() 方法

作用：比较两个对象（重写前：比较的是对象的地址值

重写后：比较的是对象的属性值）

注意：

重写前：基本数据类型比较的是内容，而引用数据类型比较的地址值。

类函数：（定义名字和年龄的标准类 + 重写 qeuals() 方法）

主函数：

public class Testequals {  
 public static void main(String[] args) {  
 String str1 = "abc";  
 String str2 = "abc";  
 // String 类中已经对 equals 方法进行了重写，直接比较内容，而非地址值  
 System.out.println(str1.equals(str2)); // true  
  
 Student stu1 = new Student("张三",23);  
 Student stu2 = new Student("张三",23);  
 /\*  
 这个为 Object 类当中默认的 equals 方法。  
 public boolean equals(Object obj) {  
 return (this == obj);  
 重点：== 比较基本数据类型的时候比较的是 内容，比较引用类型的时候比较的两个对象的地址值。  
 }\*/  
 // 没有进行重写，继承 Object类中的 equals方法 此时比较的为两个对象的 地址值。  
 System.out.println(stu1.equals(stu2)); // 重写方法前：false  
  
 // 进行重写，比较的为属性值（name age 等）+  
  
 System.out.println(stu1.equals(stu2)); // 重写方法后：true  
  
 }  
}

1. Objects类

1.equals() 方法

作用：比较两个对象是否相同，但是加了一些健壮性的判断（Objects）

类函数：标准类

主函数：

import java.util.Objects;  
  
public class TestObjects {  
 public static void main(String[] args) {  
  
 String str1 = "abc";  
 String str2 = "abc";  
 System.out.println(str1.equals(str2)); // true  
  
 String str3 = null;  
 System.out.println(str1.equals(str3)); // NullPointerException  
 boolean result = Objects.equals(str1,str2);  
 System.out.println(result); // false  
 }  
}