

- 1 Object 类型的变量
- 2 equals() 方法
- 3 hashCode() 方法
- 4 toString() 方法

Object 类是 Java 中所有类的始祖，在 Java 中每个类都扩展了 Object 类。在定义类时，如果没有明确地指出超类，Object 类就被认为是这个类的超类。下面介绍 Object 类的一些基本内容。

# 1 Object 类型的变量

可以使用 Object 类型的变量引用任何类型的对象。但 Object 类型的变量只能用于作为各种值的一个泛型容器，要想对其中的内容进行具体操作，还需要清楚对象的原始类型，并进行相应的强制类型转换。例如：

```
Object obj = new Employee("Harry", 35000);
Employee e = (Employee) obj;
```

所有的数组类型，不管是对象数组还是基本类型的数组都扩展了 Object 类。例如，下列操作是合法的：

```
Employee[] staff = new Employee[10];
Object obj = staff;
obj = new int[10];
```

# 2 equals() 方法

Object 类中的 equals() 方法用于检测一个对象是否等于另一个对象。Object 类中实现的 equals() 方法将确定两个对象引用是否相等。例如：

```
Employee a = new Employee("Harry", 35000);
Employee b = a;
Employee c = new Employee("Harry", 35000);
Employee d = new Employee("Peter", 30000);

Object object = a;
System.out.println(object.equals(b)); // true
System.out.println(object.equals(c)); // false
System.out.println(object.equals(d)); // false
```

通常情况下，需要基于状态检测对象的相等性，如果两个对象有相同的状态，才认为这两个对象是相等的。`Object` 类中实现的 `equals()` 方法显然不能满足这种需求。为此需要自己编写 `equals()` 方法的实现。例如，`Employee` 类的 `equals()` 方法实现如下：

```
1 public boolean equals(Object otherObject)
2 {
3     // 判断两个对象引用是否相同
4     if (this == otherObject) return true;
5
6     // 当 otherObject 为 null 时, 返回 false
7     if (otherObject == null) return false;
8
9     // 如果类型不一样, 二者不可能相等
10    if (getClass() != otherObject.getClass())
11        return false;
12
13    // 至此, otherObject 一定是一个非 null 的 Employee 对象
14    Employee other = (Employee) otherObject;
15
16    // 判断二者的状态是否相同
17    return Objects.equals(name, other.name)
18        && salary == other.salary
19        && Objects.equals(hireDay, other.hireDay);
20 }
```

对上面的实现做如下说明：

1. `getClass()` 方法返回一个对象所属的类。
2. 为了防备 `name` 或 `hireDay` 可能为 `null` 的情况，需要使用 `Objects.equals()` 方法。如果两个参数都为 `null`，`Objects.equals(a, b)` 返回 `true`；如果其中一个参数为 `null`，则返回 `false`；如果两个参数都不为 `null`，则调用 `a.equals(b)`。

在子类中定义 `equals()` 方法时，首先调用超类的 `equals()`，如果检测失败，对象就不可能相等。如果超类中的字段都相等，就需要比较子类中的实例字段。例如，`Manager` 类的 `equals()` 方法可以定义如下：

```
1 public boolean equals(Object otherObject)
2 {
3     if (!super.equals(otherObject)) return false;
4
5     Manager other = (Manager) otherObject;
6     return bonus == other.bonus;
7 }
```

Java 语言规范要求 `equals()` 方法具有下面的特性：

1. 自反性：对于任何非空引用 `x`，`x.equals(x)` 应该返回 `true`。
2. 对称性：对于任何引用 `x` 和 `y`，当且仅当 `y.equals(x)` 返回 `true` 时，`x.equals(y)` 返回 `true`。  
也就是说，`y.equals(x)` 和 `x.equals(y)` 返回的结果相同。
3. 传递性：对于任何引用 `x`、`y` 和 `z`，如果 `x.equals(y)` 返回 `true`，`y.equals(z)` 返回 `true`，`x.equals(z)` 也应该返回 `true`。
4. 一致性：如果 `x` 和 `y` 引用的对象没有发生变化，反复调用 `x.equals(y)` 应该返回同样的结果。
5. 对于任意非空引用 `x`，`x.equals(null)` 应该返回 `false`。

下面给出编写一个完美的 `equals()` 方法的建议：

1. 显式参数命名为 `otherObject`，稍后需要将它强制转换成另一个名为 `other` 的变量。
2. 检测 `this` 与 `otherObject` 是否相等：`if (this == otherObject) return true;`
3. 检测 `otherObject` 是否为 `null`，如果为 `null`，返回 `false`：  
`if (otherObject == null) return false;`
4. 比较 `this` 与 `otherObject` 的类。如果 `equals` 的语义可以在子类中改变，就使用 `getClass()` 检测：  
`if (getClass() != otherObject.getClass()) return false;`  
如果所有的子类都有相同的相等性语义，可以使用 `instanceof` 检测：  
`if (!(otherObject instanceof ClassName)) return false;`
5. 将 `otherObject` 强制转换为相应类类型的变量：  
`ClassName other = (ClassName) otherObject;`
6. 根据相等性概念的要求来比较字段。使用 `==` 比较基本类型字段，使用 `Objects.equals()` 比较对象字段。对于基本类型的数组字段，可以使用静态的 `Arrays.equals()` 方法检测相应的数组元素是否相等。

如果在子类中重新定义 `equals()`，就要在其中包含一个 `super.equals(otherObject)` 调用。

对于上面提到的相等性语义，做出如下说明。在 `Employee` 和 `Manager` 的例子中，只要对应的字段相等，就认为两个对象相等。如果两个 `Manager` 对象的姓名、薪水和雇用日期均相等，而奖金不相等，就认为它们是不相同的，因此要使用 `getClass()` 检测。但是，假设使用员工的ID作为相等性检测标准，并且这个相等性概念适用于所有的子类，就可以使用 `instanceof` 检测，这样可以在不同子类的对象之间进行相等性比较。

## 3 hashCode() 方法

散列码 (hash code) 是由对象导出的一个整数值。散列码是没有规律的，如果 `x` 和 `y` 是两个不同的对象，`x.hashCode()` 和 `y.hashCode()` 基本上不会相同。

由于 `hashCode()` 方法定义在 `Object` 类中，因此每个对象都有一个默认的散列码，其值由对象的存储地址得出。如果在类中没有定义 `hashCode()` 方法，将采用 `Object` 类中的默认 `hashCode()` 方法，从对象的存储地址得出散列码。

`String` 类的 `hashCode()` 方法实现如下：

```
int hash = 0;
for (int i = 0; i < length(); i++)
{
    hash = 31 * hash + charAt(i);
}
```

可见字符串的散列码是由内容导出的。

`hashCode()` 方法应该返回一个整数（可以是负数）。要合理地组合实例字段的散列码，以便能够让不同对象产生的散列码分布更加均匀。对于对象字段，使用 `null` 安全的 `Objects.hashCode(Object)` 方法，当参数为 `null` 时返回 0，否则返回对参数调用 `hashCode()` 的结果；对于基本类型字段，使用包装器的静态 `hashCode()` 方法；对于数组字段，使用静态的 `Arrays.hashCode()` 方法。例如，`Employee` 类的 `hashCode()` 方法可以实现如下：

```
public int hashCode()
{
    return 7 * Objects.hashCode(name)
        + 11 * Double.hashCode(salary)
        + 13 * Objects.hashCode(hireDay);
}
```

其中系数 7、11、13 是自己设定的，对象字段 `name` 和 `hireDay` 使用 `Objects.hashCode(Object)` 方法，`double` 类型的 `salary` 属性使用 `Double.hashCode()` 方法。

需要组合多个散列值时，可以调用 `Objects.hash()` 并将所有字段作为参数。这个方法会对各个参数调用 `Objects.hashCode()`，并组合这些散列值。例如，上面的 `Employee` 类的 `hashCode()` 方法可以写为：

```
public int hashCode()
{
    return Objects.hash(name, salary, hireDay);
}
```

`equals()` 方法与 `hashCode()` 方法的定义必须兼容，如果 `x.equals(y)` 返回 `true`，那么 `x.hashCode()` 和 `y.hashCode()` 就必须返回相同的值。如果重新定义了 `equals()` 方法，就必须为用户可能插入散列表的对象重新定义 `hashCode()` 方法。

## 4 toString() 方法

`Object` 类中的 `toString()` 方法会返回表示对象值的一个字符串。例如，`Point` 类的 `toString()` 方法会返回形如这样的字符串：`java.awt.Point[x=10,y=20]`

绝大多数的 `toString()` 方法遵循这样的格式：类名，随后是一对方括号括起来的字段值。例如，`Employee` 类的 `toString()` 方法可以实现如下：

```
public String toString()
{
    return getClass().getName()
        + "[name=" + name
        + ",salary=" + salary
        + ",hireDay=" + hireDay
        + "];"
}
```

`getClass()` 方法返回包含对象信息的类对象，`getName()` 方法返回字符串形式的类名。

设计子类时应该单独定义子类的 `toString()` 方法，并加入子类的字段。如果超类使用了 `getClass().getName()`，子类只需要调用 `super.toString()` 再加上子类字段就可以了。例如，`Manager` 类的 `toString()` 方法可以实现如下：

```
public String toString()
{
    return super.toString()
        + "[bonus=" + bonus
        + "];"
}
```

只要对象与一个字符串通过操作符 `+` 连接起来，Java 编译器就会自动调用 `toString()` 方法来获得这个对象的字符串描述。例如：

```
Point p = new Point(10, 20);
String message = "The current position is " + p;
```

如果 `x` 是一个任意对象，调用 `System.out.println(x)` 时就会自动调用 `x.toString()`，并输出得到的字符串。

`Object` 类中定义的默认的 `toString()` 方法可以得到对象的类名和散列码。例如，调用 `System.out.println(System.out)`，会输出 `java.io.PrintStream@2f6684`。之所以得到这样的结果，是因为 `PrintStream` 类没有覆盖 `toString()` 方法。

`toString()` 方法是一种非常实用的调试工具。在标准类库中，许多类都定义了 `toString()` 方法，以使用户能够获得一些有关对象状态的有用信息。建议为自定义的每一个类添加 `toString()` 方法。