# 尚马教育 JAVA 基础课程

# Lang包常用类

文档编号：A11

创建日期： 2017-04-12

最后修改日期：2019-09-23

版 本 号：V3.0

电子版文件名：尚马教育-第一阶段-11.jdk语言包专题课程.docx

**文档修改记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新作者 | 更新说明 | 版本号 |
| 2017-07-30 | 张元林 | 初始版本 | V1.0 |
| 2018-08-01 | 王绍成 | Java基础版本更新 | V2.0 |
| 2019-08-09 | 徐丽莎 | Java基础版本更新 | V3.0 |

**主讲人：**

**徐丽莎**

目录

[尚马教育 JAVA 基础课程 1](#_Toc25435)

[Lang包常用类 1](#_Toc16039)

[1. 包装类 2](#_Toc24937)

[1.1. 概念 2](#_Toc25446)

[1.2. 分类 3](#_Toc17942)

[1.3. Integer 3](#_Toc11713)

[1.3.1. 方法 3](#_Toc10876)

[1.3.2. 装箱与拆箱 3](#_Toc12357)

[1.3.3. 整数缓存池 3](#_Toc9575)

[1.4. Character 4](#_Toc828)

[1.4.1. 方法 4](#_Toc32767)

[1.4.2. 案例 4](#_Toc7461)

[2. Math 5](#_Toc18641)

[2.1. 方法 5](#_Toc17442)

[2.2. 案例 5](#_Toc31)

[3. Object 5](#_Toc1929)

[3.1. 方法 6](#_Toc12184)

[3.2. 案例 6](#_Toc27446)

[3.3. 克隆 clone 6](#_Toc4329)

[4. Class 6](#_Toc31324)

[4.1. 操作属性 7](#_Toc16841)

[4.2. 操作方法 7](#_Toc26082)

[4.3. 操作构造方法 7](#_Toc14698)

[5. 作业 7](#_Toc12936)

## 包装类

* java.lang包是java内置的一个基础包，其中包含了一系列程序中经常要用到的类；
* 在默认情况下，每个java程序都会自动导入该包，因此无需在程序中显式地声明。

### 概念

* 有基本数据类型为何又提供包装类呢？
  + 类似C++,java提供基本的数据类型将变量存储在栈里面，效率更加高效。
  + java是一门面向对象的语言，基本数据类型不具有对象的特质，为了让基本数据类型具有对象的特质，就出现了包装类。
  + 它相当于将基本类型“包装起来”，使得它具有了对象的性质，并且为其添加了属性和方法，丰富了基本类型的操作

### 分类

* 为了丰富基本数据类型(4类八种)的操作，提供8个包装类

|  |
| --- |
|  |

### Integer

* 以Integer为例，了解及掌握包装类相关的功能。
* Integer类包装一个对象中的原始类型int的值

#### 方法

|  |
| --- |
| *// 字符串转换成数字(字符串是非数字则报错)* **int** i = Integer.*parseInt*(**"123"**); System.***out***.println(**"字符串转成数字: "** + i); **int** compare = Integer.*compare*(1, 2); System.***out***.println(**"比较结果(x<y=-1 x==y=0 x>y=1) : "** + compare); **int** max = Integer.*max*(1, 2); System.***out***.println(**"最大值 : "** + max); **int** min = Integer.*min*(1, 2); System.***out***.println(**"最小值 : "** + min); **int** sum = Integer.*sum*(1, 2); System.***out***.println(**"求和 : "** + sum); String s = Integer.*toString*(123); System.***out***.println(**"数字转字符 : "** + s); Integer integer = Integer.*valueOf*(**"123"**, 10); System.***out***.println(**"valueOf"**+integer); |

#### 装箱与拆箱

|  |
| --- |
| *//包装类可以直接使用等号赋值对应的基础数据类型的值,称之为装箱* Integer i = 1; *//装箱就是JVM自动调用Integer.valueOf(int)方法,返回Integer对象* **int** i1 = i;*//拆箱就是jvm自动调用Integer对象实例的intValue方法* |

#### 整数缓存池

|  |
| --- |
| **//Integer缓存类.缓存类在JVM加载lang包的时候,自动创建一个长度为256的数组.数组中的每一个元素都是一个Integer对象.Integer对象的值范围默认为byte的数值范围**  **private static class** IntegerCache {  **static final int *low*** = -128; //缓存数组值范围的最小值  **static final int *high***; //缓存数组范围的最大值  **static final** Integer ***cache***[]; //缓存数组   **static** {  *// high value may be configured by property* **int** h = 127; //缓存数组值范围默认上限  //读取JVM的最大上限值配置  String integerCacheHighPropValue =  sun.misc.VM.*getSavedProperty*(**"java.lang.Integer.IntegerCache.high"**);  **if** (integerCacheHighPropValue != **null**) {  **try** {  //将JVM设置的参数转换成int  **int** i = *parseInt*(integerCacheHighPropValue);  //取出设置参数和127的最大值  i = Math.*max*(i, 127);  *// Maximum array size is Integer.MAX\_VALUE*  *//取出设置参数与Integer的最大上限的最小值* h = Math.*min*(i, Integer.***MAX\_VALUE*** - (-***low***) -1);  } **catch**( NumberFormatException nfe) {  *// If the property cannot be parsed into an int, ignore it.* }  }  ***high*** = h; //将最大值赋值给数组最大范围  //创建缓存数组长度(最大值--128+1)[+1的原因是因为有个数为0]  ***cache*** = **new** Integer[(***high*** - ***low***) + 1];  **int** j = ***low***;  //遍历数组长度,给每一个索引创建Integer对象,按照值范围填充  **for**(**int** k = 0; k < ***cache***.**length**; k++)  ***cache***[k] = **new** Integer(j++);   *// range [-128, 127] must be interned (JLS7 5.1.7)* **assert** IntegerCache.***high*** >= 127;  }   **private** IntegerCache() {} } |
| Integer i = 1; //默认调用Integer.valueof(int)方法  **public static** Integer valueOf(**int** i) {  //判断值是否在缓存访问内  **if** (i >= IntegerCache.***low*** && i <= IntegerCache.***high***)  //取出缓存中的对象  **return** IntegerCache.***cache***[i + (-IntegerCache.***low***)];  **return new** Integer(i); //不在直接return一个全新的Integer对象 } |

使用-XX:AutoBoxCacheMax调整缓存池大小

### Character

#### 方法

|  |
| --- |
|  |

#### 案例

|  |
| --- |
| **public** **class** CharacterDemo {  **public** **static** **void** main(String[] args) {  **char**[] charArray = { '\*', '7', 'b', ' ', 'A' };  **for** (**int** i = 0; i < charArray.length; i++) {  **if** (Character.*isDigit*(charArray[i])) {  System.***out***.println(charArray[i] + "是一个数字。");  }  **if** (Character.*isLetter*(charArray[i])) {  System.***out***.println(charArray[i] + "是一个字母。");  }  **if** (Character.*isWhitespace*(charArray[i])) {  System.***out***.println(charArray[i] + "是一个空格。");  }  **if** (Character.*isLowerCase*(charArray[i])) {  System.***out***.println(charArray[i] + "是小写形式。");  }  **if** (Character.*isUpperCase*(charArray[i])) {  System.***out***.println(charArray[i] + "是大写形式。");  }  }  }  } |

## Math

* Math类中提供了一系列基本数学运算和几何运算的方法；
* 该类的构造方法被修饰为private，因此不能实例化；
* 该类中的所有方法都是静态的，可以通过类名直接调用；
* 该类被修饰为final，因此没有子类

### 方法

|  |
| --- |
|  |

### 案例

|  |
| --- |
| System.***out***.println(**"求绝对值"** + Math.*abs*(-123)); System.***out***.println(**"求幂 : "** + Math.*pow*(10, 0)); System.***out***.println(**"平方根 : "** + Math.*sqrt*(4)); System.***out***.println(**"四舍五入 : "** + Math.*round*(4.4)); System.***out***.println(**"返回最大整数 : "** + Math.*ceil*(5.1)); System.***out***.println(**"返回最小整数 : "** + Math.*floor*(5.9));  *//工作中涉及到钱的数据类型只允许使用BigDecimal* BigDecimal bd1 = **new** BigDecimal(5.4); BigDecimal bd2 = **new** BigDecimal(5.4); BigDecimal add = bd1.add(bd2); *//加法* System.***out***.println(add.doubleValue()); |

## Object

* Java中的类体系遵循单根结构，即任何一个类往上追溯都到达同一个父类；
* Object类就是这个单根体系的根，也就是说它是其它所有类的共同父类；
* 如果用户定义的类没有扩展任何其它类，则默认扩展自Object类；
* Object类中定义的一些方法，会被继承到所有类中

### 方法

|  |
| --- |
|  |

### 案例

|  |
| --- |
| //在Java中打印对象时,默认是调用Object的toString方法,该方法是打印类名和对象的HashCode码组成的一串字符. 如果想要打印对象时直接打印对象的属性值,可以重写对象的toString方法  @Override **public** String toString() {  **return "Student{"** +  **"id="** + **id** +  **", name='"** + **name** + **'\''** +  **'}'**; }  //若对象在使用过程中需要进行比较,则需要生成对象的Hashcode和Equals方法  @Override **public boolean** equals(Object o) {  **if** (**this** == o) **return true**;  **if** (o == **null** || getClass() != o.getClass()) **return false**;  Student student = (Student) o;  **return** Objects.*equals*(**id**, student.**id**) &&  Objects.*equals*(**name**, student.**name**); }  @Override **public int** hashCode() {   **return** Objects.*hash*(**id**, **name**); }  **Hashcode相同,不代表对象相同**  **Hashcode不同,对象一定不同** |

### 克隆 clone

|  |
| --- |
| //浅拷贝  //实现Cloneable接口,并实现clone()方法  **public class** Student **implements** Cloneable{  **private** Integer **id**;  **private** String **name**;   @Override  **protected** Student clone() **throws** CloneNotSupportedException {  **return** (Student) **super**.clone();  }  } |
| //深拷贝. 当对象中包含有引用数据类型的成员属性时,在操作拷贝对象时不影响原有的对象,则需要手动将引用数据类型拷贝出来并重新赋值  **public class** Student **implements** Cloneable{  **private** Integer **id**;  **private** String **name**;  **private** String[] **hobby**;  **private** Classes **classes**;   @Override  **protected** Student clone() **throws** CloneNotSupportedException {  Student stu = (Student) **super**.clone();  *//拷贝数组* String[] newHobbys = Arrays.*copyOf*(**hobby**, **this**.**hobby**.**length**);  stu.setHobby(newHobbys);  *//拷贝对象* Classes cloneClasses = **this**.**classes**.clone();  stu.setClasses(cloneClasses);  **return** stu;  }  } |

## Class(反射)

* Java应用程序实际上都是由一个个对象组成，这些对象分别属于什么类，是从哪个类继承而来，这一系列的信息都是由Class类的实例来记录的；
* Class类的实例用于记录对象的类描述信息；
* 如果在程序运行时，需要检索某个对象的相关类信息，可以调用该对象的getClass方法来获得记录其描述信息的Class类实例；
* Class类没有公共的构造方法，无法通过new运算符实例化，只能通过对象的getClass方法，或是通过Class的静态方法forName或者是class属性来获得实例。
* 反射： 通过Class的对象 操作类或者接口的属性 方法 构造
  + 在反射里面，将类的属性封装成Field进行操作
  + 将类的方法封装成Method进行操作
  + 将类的构造封装成Constructor进行操作

### 操作属性

|  |
| --- |
|  |

### 操作方法

|  |
| --- |
|  |

### 操作构造方法

|  |
| --- |
|  |

## 作业

1. 不利用Integer.parseInt()方法将一个给定的字符串转变为Int型数值,并检测自己实现的方法和Integer.parseInt()的性能差距。
2. 编写一个Student类，包含name、age等属性，要求使用System.out.println()打印Student类的对象引用时，输出的为name的值！