**构造方法是一种特殊的方法，具有以下特点。**

**（1）构造方法的方法名必须与类名相同。**

**（2）构造方法没有返回类型，也不能定义为void，在方法名前面不声明方法类型。**

**（3）构造方法的主要作用是完成对象的初始化工作，它能够把定义对象时的参数传给对象的域。**

**（4）一个类可以定义多个构造方法，如果在定义类时没有定义构造方法，则[编译系统](https://wenwen.sogou.com/s/?w=%E7%BC%96%E8%AF%91%E7%B3%BB%E7%BB%9F&ch=ww.xqy.chain" \t "https://wenwen.sogou.com/z/_blank)会自动插入一个无参数的默认[构造器](https://wenwen.sogou.com/s/?w=%E6%9E%84%E9%80%A0%E5%99%A8&ch=ww.xqy.chain" \t "https://wenwen.sogou.com/z/_blank)，这个构造器不执行任何代码。**

**（5）构造方法可以重载，以参数的个数，类型，顺序。**

\\\1 基本类型：byte 二进制位数：8

包装类：java.lang.Byte

最小值：Byte.MIN\_VALUE=-128

最大值：Byte.MAX\_VALUE=127

基本类型：short 二进制位数：16

包装类：java.lang.Short

最小值：Short.MIN\_VALUE=-32768

最大值：Short.MAX\_VALUE=32767

基本类型：int 二进制位数：32

包装类：java.lang.Integer

最小值：Integer.MIN\_VALUE=-2147483648

最大值：Integer.MAX\_VALUE=2147483647

基本类型：long 二进制位数：64

包装类：java.lang.Long

最小值：Long.MIN\_VALUE=-9223372036854775808

最大值：Long.MAX\_VALUE=9223372036854775807

基本类型：float 二进制位数：32

包装类：java.lang.Float

最小值：Float.MIN\_VALUE=1.4E-45

最大值：Float.MAX\_VALUE=3.4028235E38

基本类型：double 二进制位数：64

包装类：java.lang.Double

最小值：Double.MIN\_VALUE=4.9E-324

最大值：Double.MAX\_VALUE=1.7976931348623157E308

基本类型：char 二进制位数：16

包装类：java.lang.Character

最小值：Character.MIN\_VALUE=0

最大值：Character.MAX\_VALUE=65535

\\\2 下表列出了 Java 各个类型的默认值：

byte 0

short 0

int 0

long 0L

float 0.0f

double 0.0d

char 'u0000'

String (or any object) null

boolean false

\\\3 byte、int、long、和short都可以用十进制、16进制以及8进制的方式来表示。

当使用字面量的时候，前缀 0 表示 8 进制，而前缀 0x 代表 16 进制, 例如：

int decimal = 100;

int octal = 0144;

int hexa = 0x64;

\\\4 Java语言支持一些特殊的转义字符序列。

符号 字符含义

\n 换行 (0x0a)

\r 回车 (0x0d)

\f 换页符(0x0c)

\b 退格 (0x08)

\0 空字符 (0x0)

\s 空格 (0x20)

\t 制表符

\" 双引号

\' 单引号

\\ 反斜杠

\ddd 八进制字符 (ddd)

\uxxxx 16进制Unicode字符 (xxxx)

\\\5 数据类型转换必须满足如下规则：

1. 不能对boolean类型进行类型转换。

2. 不能把对象类型转换成不相关类的对象。

3. 在把容量大的类型转换为容量小的类型时必须使用强制类型转换。

4. 转换过程中可能导致溢出或损失精度，例如：

int i =128;

byte b = (byte)i;

因为 byte 类型是 8 位，最大值为127，所以当 int 强制转换为 byte 类型时，值 128 时候就会导致溢出。

\\\6 包装类 基本数据类型

Boolean boolean

Byte byte

Short short

Integer int

Long long

Character char

Float float

Double double

\\\7 Java 的 Math 包含了用于执行基本数学运算的属性和方法，如初等指数、对数、平方根和三角函数。

Math 的方法都被定义为 static 形式，通过 Math 类可以在主函数中直接调用。

("90 度的正弦值：" + Math.sin(Math.PI/2)

("0度的余弦值：" + Math.cos(0)

("60度的正切值：" + Math.tan(Math.PI/3)

("1的反正切值： " + Math.atan(1)

("π/2的角度值：" + Math.toDegrees(Math.PI/2)

\\\8 Number & Math 类方法

下面的表中列出的是 Number & Math 类常用的一些方法：

序号 方法与描述

1 xxxValue()

将 Number 对象转换为xxx数据类型的值并返回。

2 compareTo()

将number对象与参数比较。

3 equals()

判断number对象是否与参数相等。

4 valueOf()

返回一个 Number 对象指定的内置数据类型

5 toString()

以字符串形式返回值。

6 parseInt()

将字符串解析为int类型。

7 abs()

返回参数的绝对值。

8 ceil()

返回大于等于( >= )给定参数的的最小整数，类型为双精度浮点型。

9 floor()

返回小于等于（<=）给定参数的最大整数 。

10 rint()

返回与参数最接近的整数。返回类型为double。

11 round()

它表示四舍五入，算法为 Math.floor(x+0.5)，即将原来的数字加上 0.5 后再向下取整，所以，Math.round(11.5) 的结果为12，Math.round(-11.5) 的结果为-11。

12 min()

返回两个参数中的最小值。

13 max()

返回两个参数中的最大值。

14 exp()

返回自然数底数e的参数次方。

15 log()

返回参数的自然数底数的对数值。

16 pow()

返回第一个参数的第二个参数次方。

17 sqrt()

求参数的算术平方根。

18 sin()

求指定double类型参数的正弦值。

19 cos()

求指定double类型参数的余弦值。

20 tan()

求指定double类型参数的正切值。

21 asin()

求指定double类型参数的反正弦值。

22 acos()

求指定double类型参数的反余弦值。

23 atan()

求指定double类型参数的反正切值。

24 atan2()

将笛卡尔坐标转换为极坐标，并返回极坐标的角度值。

25 toDegrees()

将参数转化为角度。

26 toRadians()

将角度转换为弧度。

27 random()

返回一个随机数。

Math 的 floor,round 和 ceil 方法实例比较

参数 Math.floor Math.round Math.ceil

1.4 1 1 2

1.5 1 2 2

1.6 1 2 2

-1.4 -2 -1 -1

-1.5 -2 -1 -1

-1.6 -2 -2 -1

\\\9 String 类是不可改变的，所以你一旦创建了 String 对象，那它的值就无法改变了

\\\10 我们知道输出格式化数字可以使用 printf() 和 format() 方法。

String 类使用静态方法 format() 返回一个String 对象而不是 PrintStream 对象。

String 类的静态方法 format() 能用来创建可复用的格式化字符串，而不仅仅是用于一次打印输出。

如下所示：

System.out.printf("浮点型变量的值为 " +

"%f, 整型变量的值为 " +

" %d, 字符串变量的值为 " +

"is %s", floatVar, intVar, stringVar);

你也可以这样写

String fs;

fs = String.format("浮点型变量的值为 " +

"%f, 整型变量的值为 " +

" %d, 字符串变量的值为 " +

" %s", floatVar, intVar, stringVar);

\\\11 StringBuffer 和 StringBuilder 类

当对字符串进行修改的时候，需要使用 StringBuffer 和 StringBuilder 类。

和 String 类不同的是，StringBuffer 和 StringBuilder 类的对象能够被多次的修改，并且不产生新的未使用对象。

StringBuilder 类在 Java 5 中被提出，它和 StringBuffer 之间的最大不同在于 StringBuilder 的方法不是线程安全的（不能同步访问）。

由于 StringBuilder 相较于 StringBuffer 有速度优势，所以多数情况下建议使用 StringBuilder 类。然而在应用程序要求线程安全的情况下，则必须使用 StringBuffer 类。

public class Test{

public static void main(String args[]){

StringBuffer sBuffer = new StringBuffer("菜鸟教程官网：");

sBuffer.append("www");

sBuffer.append(".runoob");

sBuffer.append(".com");

System.out.println(sBuffer);

}

}

以上实例编译运行结果如下：

菜鸟教程官网：www.runoob.com

数组

\\\12

数组的元素是通过索引访问的。数组索引从 0 开始，所以索引值从 0 到 arrayRefVar.length-1。

\\\13

实例

一、下面的语句首先声明了一个数组变量 myList，接着创建了一个包含 10 个 double 类型元素的数组，并且把它的引用赋值给 myList 变量。

TestArray.java 文件代码：

public class TestArray {

public static void main(String[] args) {

// 数组大小

int size = 10;

// 定义数组

double[] myList = new double[size];

myList[0] = 5.6;

myList[1] = 4.5;

myList[2] = 3.3;

myList[3] = 13.2;

myList[4] = 4.0;

myList[5] = 34.33;

myList[6] = 34.0;

myList[7] = 45.45;

myList[8] = 99.993;

myList[9] = 11123;

// 计算所有元素的总和

double total = 0;

for (int i = 0; i < size; i++) {

total += myList[i];

}

System.out.println("总和为： " + total);

}

}

以上实例输出结果为：

总和为： 11367.373

二、TestArray.java 文件代码：

public class TestArray {

public static void main(String[] args) {

double[] myList = {1.9, 2.9, 3.4, 3.5};

// 打印所有数组元素

for (int i = 0; i < myList.length; i++) {

System.out.println(myList[i] + " ");

}

// 计算所有元素的总和

double total = 0;

for (int i = 0; i < myList.length; i++) {

total += myList[i];

}

System.out.println("Total is " + total);

// 查找最大元素

double max = myList[0];

for (int i = 1; i < myList.length; i++) {

if (myList[i] > max) max = myList[i];

}

System.out.println("Max is " + max);

}

}

以上实例编译运行结果如下：

1.9

2.9

3.4

3.5

Total is 11.7

Max is 3.5

\\\14

For-Each 循环

JDK 1.5 引进了一种新的循环类型，被称为 For-Each 循环或者加强型循环，它能在不使用下标的情况下遍历数组。

语法格式如下：

for(type element: array)

{

System.out.println(element);

}

实例

该实例用来显示数组 myList 中的所有元素：

TestArray.java 文件代码：

public class TestArray {

public static void main(String[] args) {

double[] myList = {1.9, 2.9, 3.4, 3.5};

// 打印所有数组元素

for (double element: myList) {

System.out.println(element);

}

}

}

以上实例编译运行结果如下：

1.9

2.9

3.4

3.5