# 工具类

工具类就是为一些常用场景去设置的，比如比较大小，求平方等等

## 一、数学工具类

| 方法名 | 描述 | 示例 |
| --- | --- | --- |
| `abs()` | 计算并返回参数的绝对值 | `int absolute = Math.abs(-10); // 结果：10` |
| `ceil()` | 向上取整，返回大于或等于参数的最小整数 | `double ceiling = Math.ceil(3.½); // 结果：4.0` |
| `floor()` | 向下取整，返回小于或等于参数的最大整数 | `double floorValue = Math.floor(3.7); // 结果：3.0` |
| `round()` | 四舍五入取整，返回最接近参数的整数 | `int rounded = Math.round(3.¼); // 结果：3` |
| `sqrt()` | 计算并返回参数的平方根 | `double root = Math.sqrt(25); // 结果：5.0` |
| `pow()` | 返回参数a的b次幂 | `double power = Math.pow(2, 3); // 结果：8.0` |
| `max()` | 返回两个参数中的较大值 | `int maxNum = Math.max(10, 20); // 结果：20` |
| `min()` | 返回两个参数中的较小值 | `int minNum = Math.min(5, -5); // 结果：-5` |
| `PI` | π (圆周率) 的常数值 | `double pi = Math.PI; // 结果：约3.141592653589793` |
| `E` | 自然对数的底数e的常数值 | `double e = Math.E; // 结果：约2.718281828459045` |
| `sin()` | 计算并返回参数的正弦值（参数为弧度） | `double sine = Math.sin(Math.PI / 2); // 结果：1.0` |
| `asin()` | 计算并返回参数的反正弦值（结果为弧度） | `double arcsine = Math.asin(0.5); // 结果：约0.5235987755982988` |
| `cos()` | 计算并返回参数的余弦值（参数为弧度） | `double cosine = Math.cos(Math.PI); // 结果：-1.0` |
| `acos()` | 计算并返回参数的反余弦值（结果为弧度） | `double arccosine = Math.acos(1.0); // 结果：0.0` |
| `tan()` | 计算并返回参数的正切值（参数为弧度） | `double tangent = Math.tan(Math.PI / 4); // 结果：1.0` |
| `log()` | 计算并返回参数的自然对数（以e为底） | `double naturalLog = Math.log(Math.E); // 结果：1.0` |

| 方法名 | 描述 | 示例 |
| --- | --- | --- |
| `exp()` | 计算并返回e的参数次幂 | `double exponent = Math.exp(1.0); // 结果: 2.718281828459045` |

注意：

- 所有涉及角度的操作，`Math`类中的方法默认接收弧度作为参数。若要处理角度，通常需要先将角度转换为弧度（使用`toRadians()`方法）或将弧度转换为角度（使用`toDegrees()`方法）。
- 示例中的部分结果为近似值，实际使用时会得到更精确的结果。

1️⃣ `JAVA`不仅仅只提供加减乘除的运算符，还提供了进行高阶数学的数学工具，要使用数学工具就需要使用到`Match`类它也是`java.lang`下的包

```
public class Main {
    public static void main(String[] args)  {
        System.out.println(Math.pow(2, 10)); //求次方
        System.out.println(Math.abs(-10)); //计算绝对值
        System.out.println(Math.sqrt(6)); //求算术平方根

        /*
        输出:
        1024.0
        10
        2.449489742783178
         */
    }
}

```

2️⃣ 它还可以计算三角函数`Math.PI`表示Π

```
     public static void main(String[] args)  {
        System.out.println(Math.sin(Math.PI / 2));      //求π/2的正弦值，这里我们可以使用预置的PI进行计算
        System.out.println(Math.cos(Math.PI));        //求π的余弦值
        System.out.println(Math.tan(Math.PI / 4));     //求π/4的正切值

        System.out.println(Math.asin(1));      //三角函数的反函数也是有的，这里是求arcsin1的值
        System.out.println(Math.acos(1));
        System.out.println(Math.atan(0));
        /*
        输出
        1.0
        -1.0
        0.9999999999999999
        1.5707963267948966
        0.0
        0.0
         */
```

```
18      }
```

有时候出现的数值可能是科学计数法比如求`Sinπ`的结果


结果应该为0才对，这个E表示10，这个段句子表示

$$1.2246467991473532\times 10\text{的负}16\text{次幂}$$

---

3 也可以计算对数函数

```
public static void main(String[] args) {
    Math.log(Math.E);    //e为底的对数函数，其实就是ln，我们可以直接使用Math中定义好的e
    Math.log10(100);     //10为底的对数函数
    //利用换底公式，我们可以弄出来任何我们想求的对数函数
    double a = Math.log(4) / Math.log(2);    //这里是求以2为底4的对数，log(2)4 = ln4 / ln2
    System.out.println(a);
}
```

4 向上向下取整

```
public static void main(String[] args) {
    Math.ceil(4.5);    //通过使用ceil来向上取整
    Math.floor(5.6);   //通过使用floor来向下取整
}
```

5 `Random`随机数，这个和`Python`的类似，但是`Random`类生成的随机数是伪随机数，即它们是由确定性算法产生的看似随机的序列。相同种子值创建的`Random`对象，对于相同的调用序列，将生成相同的随机数序列。

程序中的随机并不是真正的随机，而是根据某些东西计算出来的

```
    public static void main(String[] args)  {
        Random random  = new Random();
        for (int i = 0; i < 30; i++) {
            System.out.print(random.nextInt(100) + " ");

            /*
            输出:
            95 96 59 2 28 60 11 76 48 88 54 97 53 27 78 59 40 14 36 0 20 27 62 8
8 14 15 8 67 74
             */
        }
    }
```

| 方法名 | 描述 | 示例 |
| --- | --- | --- |
| `Random()` | 默认构造函数，使用当前时间作为种子创建一个`Random`实例 | `Random rand = new Random();` |
| `Random(long seed)` | 使用指定的`long`类型种子创建一个`Random`实例 | `Random rand = new Random(12345L);` |
| `nextBoolean()` | 返回一个`boolean`值，true或false，概率相等 | `boolean result = rand.nextBoolean();` |
| `nextInt()` | 返回一个随机的`int`值，范围在`Integer.MIN_VALUE`到`Integer.MAX_VALUE`之间 | `int randomInt = rand.nextInt();` |
| `nextInt(int bound)` | 返回一个随机的`int`值，范围在[0, bound)之间（包括0，不包括bound） | `int boundedInt = rand.nextInt(100); // 生成0到99之间的整数` |
| `nextLong()` | 返回一个随机的`long`值，范围在`Long.MIN_VALUE`到`Long.MAX_VALUE`之间 | `long randomLong = rand.nextLong();` |
| `nextFloat()` | 返回一个随机的`float`值，范围在[0.0f, 1.0f)之间（包括0.0f，不包括1.0f） | `float randomFloat = rand.nextFloat();` |
| `nextDouble()` | 返回一个随机的`double`值，范围在[0.0, 1.0)之间（包括0.0，不包括1.0） | `double randomDouble = rand.nextDouble();` |
| `nextBytes(byte[] bytes)` | 将指定长度的字节数组填充值，每个元素为[0, 255]范围内的随机字节 | `byte[] randomBytes = new byte[10]; rand.nextBytes(randomBytes);` |

## 二、数组工具类

基本常用:

以下是Arrays工具类的一些常用方法及其用法:

| 方法名 | 描述 | 示例代码 |
| --- | --- | --- |
| `asList()` | 将数组转换为List集合 | `List<String> list = Arrays.asList("A", "B", "C");` |
| `binarySearch()` | 对已排序的数组进行二分查找，返回元素索引 | `int index = Arrays.binarySearch(arr, target);` |
| `copyOf()` | 复制数组，并指定新数组的长度 | `int[] newArr = Arrays.copyOf(arr, newLength);` |
| `copyOfRange()` | 复制数组的一部分，并指定新数组的长度 | `int[] newArr = Arrays.copyOfRange(arr, startIndex, endIndex);` |
| `fill()` | 用指定的值填充数组 | `Arrays.fill(arr, value);` |
| `equals()` | 比较两个数组是否相等 | `boolean isEqual = Arrays.equals(arr1, arr2);` |
| `deepEquals()` | 比较两个多维数组是否相等 | `boolean isDeepEqual = Arrays.deepEquals(arr1, arr2);` |
| `sort()` | 对数组进行排序 | `Arrays.sort(arr);` |
| `sort(Comparator)` | 根据指定的比较器对数组进行排序 | `Arrays.sort(arr, comparator);` |
| `toString()` | 将数组转换为字符串表示形式 | `String str = Arrays.toString(arr);` |
| `stream()` | 将数组转换为Stream流 | `IntStream stream = Arrays.stream(arr);` |

这些方法都是java.util.Arrays工具类中提供的，可以方便地对数组进行操作。

数据工具类可以使得我们操作数组更加便捷，比如我想要打印一个数组，就可以没必要遍历这个数组，而是直接使用数组工具类来搞定，数据工具类名为`Arrays`,它也是`java.lang`下的包

1. 印数组，可以直接通过`toString`方法

```
    public static void main(String[] args)  {
        int[] arr = {1,2,3,4,5,6};
        System.out.println(Arrays.toString(arr));
        /*
        输出:
        [1, 2, 3, 4, 5, 6]
        */
    }
```

2. 还可以对数组排序使用`sort`

```
    public static void main(String[] args)  {
        int[] arr = {1,2,4,3,5,6};
        Arrays.sort(arr);
        System.out.println(Arrays.toString(arr));
        //输出:[1, 2, 3, 4, 5, 6]
    }
```

3. 填充数组，使用`fill`

```
    public static void main(String[] args)  {
        int[] arr = new int[10];
        Arrays.fill(arr,10);
        System.out.println(Arrays.toString(arr));
        //输出: [10, 10, 10, 10, 10, 10, 10, 10, 10, 10]
    }
```

当然也可以指定数组的范围来填充

```
    public static void main(String[] args)  {
        int[] arr = new int[10];
        Arrays.fill(arr,4,8,10); //表示填充数组索引4到8的范围
        System.out.println(Arrays.toString(arr));
        //输出: [0, 0, 0, 0, 10, 10, 10, 10, 0, 0]
    }
```

4. 拷贝数组，使用`copyOf`

```
public static void main(String[] args)  {
        int[] arr = {1,2,3,4,5};
        int[] arr2 = Arrays.copyOf(arr,6); //前面为需要拷贝的数组对象，后面为新数组的长度
        System.out.println(Arrays.toString(arr));
        System.out.println(arr2.length);
    }
```

还可以只拷贝数组的内容

```
    public static void main(String[] args)  {
        int[] arr = {1,2,3,4,5};
        int[] arr2 = new int[10];
        System.arraycopy(arr,0,arr2,0,5);
        System.out.println(Arrays.toString(arr2));
    }
```

5. 如果是一个有序数组，如果想要快速找到对应的元素在哪个位置，就可以直接使用二分搜索法

```
    public static void main(String[] args)  {
        int[] arr = {1,2,3,4,5};
        System.out.println(Arrays.binarySearch(arr,5));
    }
```

6. 多维数组

```
public static void main(String[] args) {
    int[][] a = new int[][]{{2, 8, 4, 1}, {9, 2, 0, 3}};
    int[][] b = new int[][]{{2, 8, 4, 1}, {9, 2, 0, 3}};
    System.out.println(Arrays.equals(a, b));   //equals仅适用于一维数组
    System.out.println(Arrays.deepEquals(a, b));   //对于多维数组，需要使用deepEquals来进行深层次判断
}
```