

1. Collections类
2. Arrays类
3. Math类
4. BigInteger类
5. Random类
6. Scanner类
7. System.out

实用工具类

注意：当数组元素或容器被持有对象是自定义数据类型时，当进行顺序比较操作时要求其实现 `Comparable` 接口，或提供比较器；当进行等价性（如：`equals`）比较操作时，要求重写正确的 `equals` 方法。

1. Collections类

提供了一系列静态方法，用于操作 collection 接口的实现类。

按方法名的字典序排列。

```

1 int binarySearch(List<? extends Comparable<? super T>> list, T key) //
   在List中使用二分查找目标值key的索引，要求List被持有对象实现Comparable接口，且List必须已经升序排序。
2 int binarySearch(List<? extends T> list, T key, Comparator<? super T> c)
   //在List中使用二分查找目标值key的索引(比较时根据制定的比较器产生比较结果，若c为null则按被持有对象的自然顺序排序，此时要求元素实现Comparable接口根据compare方法比较)，要求必须已经升序排序。
3 void copy(List<? super T> dest, List<? extends T> src) //将所
   有元素从src列表复制到dest列表中
4 void fill(List<? super T> list,T obj) //使用指
   定元素替换指定List中的所有元素，线性时间运行。
5 int frequency(Collection<?> c, Object o) //返回
   指定 Collection 中等于指定对象的元素数。注意：通过被查找对象的equals方法进行相等比较。
6 T max(Collection<? extends T> coll) //根据
   其被持有对象的自然顺序返回给定Collection的最大元素。
7 T max(Collection<? extends T> coll, Comparator<? super T> comp) //根
   据指定比较器产生的顺序，返回给定 Collection 的最大元素。
8 T min(Collection<? extends T> coll) //根据
   其被持有对象的自然顺序返回给定Collection的最小元素。

```

```

9  T min(Collection<? extends T> coll, Comparator<? super T> comp)           //根据指定比较器产生的顺序, 返回给定 Collection 的最小元素。
10 boolean replaceAll(List<T> list, T oldVal, T newVal)                     //将指定List中的所有oldVal替换为newVal
11 void reverse(List<?> list)                                               //反转指定List中元素的顺序。
12 Comparator<T> reverseOrder()                                           //返回一个比较器, 它产生与实现了Comparable接口对象自然顺序相反的排序效果
13 Comparator<T> reverseOrder(Comparator<T> cmp)                         //返回一个比较器, 它产生与实现了与参数比较器相反的排序效果
14 void sort(List<T extends Comparable<? super T>> list)                  //根据被持有对象的自然顺序对指定List按升序进行排序。要求被持有对象实现Comparable接口。
15 void sort(List<T> list, Comparator<? super T> c)                       //根据指定的比较器 (若c为null则按被持有对象的自然顺序排序, 此时要求元素实现Comparable接口根据compare方法比较) 引起的顺序对指定的List进行排序。
16 void swap(List<?> list, int i, int j)                                  //交换指定List中指定位置的元素。

```

示例：对学生成绩信息的排序比较，查找

```

1
2  public class TestMain {
3
4      //定义为静态类, 为了能在main函数中直接调用
5      static class Student {
6          String name;
7          float english, math, total;
8
9          public Student(String n, float e, float m) {
10              this.name = n;
11              this.english = e;
12              this.math = m;
13              this.total = m + e;
14          }
15          /*
16           * @Description TODO 重载toString方法输出个人成绩信息
17           * @Date 11:52 2019/2/21
18           */
19
20          @Override
21          public String toString() {
22              return "Name:" + name + "\ntotal:" +
23                  total + "\nEnglish:" +
24                  english + "\nMath:" + math + "\n";
25          }

```

```

26
27     /*
28      * @Description TODO 重载equals方法，是为了二分查找时进行等价性比较
29      * @Date 11:52 2019/2/21
30      * @param null
31      * @return : null
32      */
33
34     @Override
35     public boolean equals(Object obj) {
36         Student oth=(Student)obj;    //强制类型转换
37         return this.name.equals(oth.name)&&
38             this.english==oth.english&&
39             this.math==oth.math;
40     }
41 }
42 /*
43  * @Description TODO 自定义比较器，实现Student对象首先按总成绩从低到高排序，
44  * 其次按名字排序（从小到大）。
45  * @Date 11:38 2019/2/21
46  */
47 static Comparator<Student> comparator = new Comparator<Student>() {
48     @Override
49     public int compare(Student o1, Student o2) {
50         if (o1.total == o2.total) {
51             return o1.name.compareTo(o2.name);
52         }
53         //乘以-1是为了总成绩的从低到高排序
54         return Float.compare(o1.total, o2.total)*-1;
55     }
56 };
57
58 public static void main(String[] args) {
59     List<Student> students = new ArrayList<>();
60     students.add(new Student("Tom", 75, 80));
61     students.add(new Student("Tim", 85, 82));
62     students.add(new Student("Bob", 95, 60));
63     students.add(new Student("Alic", 63.5F, 88));
64     students.add(new Student("Candy", 75, 80));
65
66     System.out.println("after sorting:");
67     //使用自定义比较器排序
68     Collections.sort(students, comparator);
69     for(Student stu:students){
70         System.out.println(stu.toString());

```

```

71     }
72     System.out.print("The rank of Tom: ");
73     System.out.println(Collections.binarySearch(students,new
Student("Tom", 75, 80),comparator)+1);
74     }
75 }
76 /*Output
77 after sorting:
78 Name:Tim
79 total:167.0
80 English:85.0
81 Math:82.0
82
83 Name:Bob
84 total:155.0
85 English:95.0
86 Math:60.0
87
88 Name:Candy
89 total:155.0
90 English:75.0
91 Math:80.0
92
93 Name:Tom
94 total:155.0
95 English:75.0
96 Math:80.0
97
98 Name:Alic
99 total:151.5
10 English:63.5
10 Math:88.0
10
10 The rank of Tom: 4
10 */

```

2. Arrays类

提供一系列静态方法，支持对数组的操作。

方法按字典序排序。Arrays 重载了对不同数据类型（基本数据类型、自定义数据类型）的方法，为了减少篇幅，若没有特殊区别，用泛型参数（或 Object）代替所有的数据类型；否则用 int 类型参数代表基本数据类型，用泛型参数（或 Object）代表自定义数据类型。

```

1  List<T> asList(T... a)                                //返回一个ArrayList对
    象，并将所有参数添加到该ArrayList对象中。
2
3
4  //二分查找，Arrays重载了针对不同类型的二分查找算法。要求数组元素必须已经升序排序
5  int binarySearch(int[] a, int key)                    //返回：二分查找数组a中
    key所在的下标
6  int binarySearch(int[] a, int fromIndex, int toIndex, int key) //返回：二
    分查找数组a下标范围[fromIndex,toIndex)中key所在的下标。
7  //针对自定义数据类型
8  int binarySearch(Object[] a, Object key)              //返回：二分查找数组a中
    key所在的下标。要求a中元素实现Comparable接口
9  int binarySearch(Object[] a, int fromIndex, int toIndex, Object key) //返
    回：二分查找数组a下标范围[fromIndex,toIndex)中key所在的下标。要求a中元素实现
    Comparable接口
10 int binarySearch(T[] a, T key, Comparator<? super T> c) //返回：二分查找数组a
    中key所在的下标。comparator指定比较方式，若c为null则根据元素的自然顺序比较（要求元素
    实现Comparable接口根据compare方法比较）。同样适用于基本数据类型（根据自动包装机制）
11 int binarySearch(T[] a, int fromIndex, int toIndex, T key, Comparator<?
    super T> c) //返回：二分查找数组a下标范围[fromIndex,toIndex)中key所在的下标。
    comparator指定比较方式，若c为null则根据元素的自然顺序比较（要求元素实现Comparable接
    口根据compare方法比较）。同样适用于基本数据类型（根据自动包装机制）
12
13
14 //数组复制，Arrays提供了长度复制CopyOf和范围复制CopyOfRange，当原数组长度不足时会用
    null填充，对于byte,short,int,long,float,double类型用0填充。
15 T[] copyOf(T[] original, int newLength)              //复制指定的数组，截取或用
    null 填充（如有必要），以使副本具有指定的长度。
16 T[] copyOfRange(T[] original, int from, int to)      //将指定数组的指定范围复制
    到一个新数组。
17
18
19 //等价性比较
20 boolean equals(Object[] a, Object[] a2)              //如果两个指定的数组对象彼
    此 完全相同(调用数组元素的equals方法)，则返回 true 。
21
22
23 //数组填充，Arrays提供了全部填充和范围填充
24 void fill(int[] a, int val)                          //将指定的 int 值分配给指定
    int 型数组的每个元素。
25 void fill(int[] a, int fromIndex, int toIndex, int val) //将指定的 int
    值分配给指定 int 型数组指定范围中的每个元素
26 void fill(Object[] a, Object val)                    // 将指定的 Object 引用分配给
    指定 Object 数组的每个元素。

```

```

27 void fill(Object[] a, int fromIndex, int toIndex, Object val) //将指定的
    Object 引用分配给指定 Object 数组指定范围中的每个元素。
28
29
30 //数组排序, Arrays提供了全部排序和范围排序
31 void sort(int[] a) //对指定的 int 型数组按数字升序进
    行排序。
32 void sort(int[] a, int fromIndex, int toIndex) //对指定 int 型数组的指定范
    围按数字升序进行排序。
33 void sort(Object[] a) //根据元素的自然顺序对指定对象
    数组按升序进行排序。要求数组元素实现Comparable接口
34 void sort(Object[] a, int fromIndex, int toIndex) //根据元素的自然顺序对指
    定对象数组的指定范围按升序进行排序。要求数组元素实现Comparable接口
35 void sort(T[] a, Comparator<? super T> c) // 根据指定比较器产生的顺
    序对指定对象数组进行排序。同样适用于基本数据类型（根据自动包装机制）
36 void sort(T[] a, int fromIndex, int toIndex, Comparator<? super T> c)
    //根据指定比较器产生的顺序对指定对象数组的指定范围进行排序。同样适用于基本数据类型（根据
    自动包装机制）
37
38
39 //将数组全部做表达式操作
40 void setAll(T[] array, IntFunction<? extends T> generator) //对数组每一项进
    行generator.apply(int i)操作。IntFunction是一个函数式接口
41 String toString(Object[] a) //返回指定数组内容的
    字符串表示形式。

```

示例：对数组的操作

```

1 public static void main(String[] args) {
2     int arr[] = new int[]{4, 213, 1, 5, 3, 11, 35};
3     int arr2[] = Arrays.copyOf(arr, arr.length);
4     System.out.println("whether arr equals arr2 or not:
    "+Arrays.equals(arr, arr2));
5     Arrays.sort(arr);
6     //输出排序后arr数组内的内容
7     System.out.println("after sorting arr: ");
8     System.out.println(Arrays.toString(arr));
9
10    System.out.println("show how to use setAll:");
11    System.out.print("the init state of arr2:");
12    System.out.println(Arrays.toString(arr2));
13    System.out.print("let's double every term in the arr2:");
14    Arrays.setAll(arr2, new IntUnaryOperator() {
15        //匿名表达式实现接口对象
16        @Override

```

```

17         public int applyAsInt(int operand) {
18             return arr2[operand]*2;
19         }
20     });
21     System.out.println(Arrays.toString(arr2));
22 }

```

3. Math类

Math类提供了一组静态方法支持基本数学函数的方法。

```

1 //两个常量
2 double E //底数，是所有自然数的底数。
3 double PI //π，圆周率。

```

```

1 //常用方法，Java重载了对于多种基本数据类型的方法，为了减少篇幅，去除参数类型信息，由于
  Java隐式转换机制，因此即使参数类型未double, int, long的整数类型也可以。
2
3 abs(x) //绝对值函数
4 //指数函数。返回值都是浮点数。
5 sqrt(x) //计算平方根
6 cbrt(x) //计算立方根
7 hypot(x,y) //计算 (x的平方+y的平方)的平方根
8 exp(x) //计算E^x值。
9 pow(x,y) //计算x^y
10
11 //对数函数，返回浮点数
12 log10(x) //以10为底x的对数
13 log(x) //以E为底x的对数
14
15 //三角函数，
16 cos(x) //余弦函数
17 acos(x) //反余弦函数。返回的角度范围在 0.0 到 pi 之间。
18 sin(x) //正弦函数
19 asin(x) //反正弦函数。返回的角度范围在 -pi/2 到 pi/2 之间。
20 tan(x) //正切函数
21 atan(x) //反正切函数。返回的角度范围在 -pi/2 到 pi/2 之间。
22 toDegrees(double angrad) //将用弧度表示的角转换为近似相等的用角度表示的角。
23 toRadians(double angdeg) //将用角度表示的角转换为近似相等的用弧度表示的角。
24 //舍入函数，返回的都是浮点数
25 floor(x) //下取整函数
26 ceil(x) //上取整函数
27 rint(x) //返回最接近参数的整数(转化为浮点数)，如果有2个数同样接近，则返回偶数的那个。

```

```

28
29 round(double x)          //返回对参数 x 四舍五入后所得的整数近似值。参数是float返回int,
                             参数是double返回long。
30 /*
31 1、参数的小数点后第一位<5, 运算结果为参数整数部分。
32 2、参数的小数点后第一位>5, 运算结果为参数整数部分绝对值+1, 符号（即正负）不变。
33 3、参数的小数点后第一位=5, 正数运算结果为整数部分+1, 负数运算结果为整数部分。
34 */
35
36
37 //最大值函数
38 max(x,y)                  //最大值函数
39 min(x,y)                  //最小值函数
40
41
42 //随机数
43 random()                  //返回一个随机值属于[0.0,1.0)

```

对数运算：

换底公式： $\log_a^b = \frac{\log_c^a}{\log_c^b}$

$\log_a^{(b \cdot c)} = \log_a^b + \log_a^c$

$\log_a^{\frac{b}{c}} = \log_a^b - \log_a^c$

$\log_a^{b^c} = c \cdot \log_a^b$

4. BigInteger类

BigInteger 是Java对大整数运算支持的封装类， BigInteger 对象可以表示属于 $[-2^{2147483647 \cdot 32 - 1}, 2^{2147483647 \cdot 32 - 1} - 1]$ 的整数

BigInteger 提供所有 Java 的基本整数操作符的对应物，并提供 java.lang.Math 的所有相关方法。另外， BigInteger 还提供以下运算：模算术、GCD 计算、质数测试、素数生成、位操作以及一些其他操作。

常量

```

1 static BigInteger ONE          //BigInteger常数1。
2 static BigInteger TEN          //BigInteger常数十。
3 static BigInteger ZERO         //BigInteger常数零。

```

构造函数


```

1 BigInteger(String val) //将 BigInteger 的十进制字符串表示形式转换为
  BigInteger。
2 BigInteger(String val, int radix) //将指定基数的 BigInteger 的字符串表示形式
  转换为 BigInteger。

```

常用方法

```

1 //this指调用方法的BigInteger对象。
2 BigInteger add(BigInteger val) //返回其值为 (this + val) 的
  BigInteger
3 BigInteger subtract(BigInteger val) //返回其值为 (this - val) 的
  BigInteger。
4 BigInteger multiply(BigInteger val) //返回其值为 (this * val) 的
  BigInteger
5 BigInteger divide(BigInteger val) //返回其值为 (this / val) 的
  BigInteger
6 BigInteger mod(BigInteger m) //返回其值为 (this mod m) 的
  BigInteger。
7 BigInteger pow(int exponent) //返回其值为 (thisexponent) 的
  BigInteger
8 BigInteger modPow(BigInteger exponent, BigInteger m) //返回其值为
  (thisexponent mod m) 的 BigInteger。
9 BigInteger negate() //返回其值是 (-this) 的
  BigInteger
10 BigInteger abs() //返回其值是此 BigInteger 的绝对值
  的 BigInteger。
11 BigInteger gcd(BigInteger val) //返回一个 BigInteger, 其值是
  abs(this) 和 abs(val) 的最大公约数。
12 BigInteger and(BigInteger val) //返回其值为 (this & val) 的
  BigInteger。还有or、not、xor、shiftLeft、shiftRight等位操作
13 int compareTo(BigInteger val) //将此 this与val 进行比较。this
  < val返回-1, this==val返回0,this>val返回1
14 boolean equals(BigInteger val) //判断this是否与val相等
15 BigInteger max(BigInteger val) //返回此this与val。较大的一个
16 BigInteger min(BigInteger val) //返回此this与val。较小的一个
17 double doubleValue() //将此 BigInteger 转换为 double。
  还有intValue、longValue、floatValue
18 String toString() //返回此 BigInteger 的十进制字符串
  表示形式。
19 String toString(int radix) //返回此 BigInteger 的给定基数的字
  符串表示形式。radix
20 byte[] toByteArray() //返回一个包含此BigInteger的二进制补
  码表示的字节数组。
21 BigInteger nextProbablePrime() //返回一个大于this的BigInteger,
  它是合数的概率不超过2的负100次方

```

```
22 | BigInteger probablePrime(int bitLength, Random rnd)    //返回一个指定二进制
    | 长度的正数BigInteger，它是合数的概率不超过2的负100次方，rnd是一个Random对象
```

5. Random类

Random 提供了产生伪随机数的方法，常用于随机算法和一些程序测试。

Random 类产生随机数的算法是一种伪随机算法，在进行产生随机数时，随机算法通过起源数字（seed）或叫种子数进行一系列的变换，产生随机数。**所以相同种子数在相同次数产生的随机数是相同的。**

构造方法

```
1 | public Random()    //该构造方法会使用系统当前时间的相关数字作为产生随机数的
  | 种子数并利用该种子数产生随机数；
2 | public Random(long seed)    //设置种子
```

```
1 | public boolean nextBoolean()
2 | //生成一个随机的boolean值，生成true和false的值几率相等
3 |
4 | public double nextDouble()
5 | //生成一个随机的double值，数值介于[0,1.0)之间。
6 |
7 | public double nextFloat()
8 | //生成一个随机的Float值，数值介于[0,1.0)之间。
9 |
10 | public int nextInt()
11 | //生成一个随机的int值，该值介于int的区间，也就是-2^31到2^31-1之间。
12 |
13 | public int nextInt(int n)
14 | //生成一个随机的int值，该值介于[0,n)的区间，也就是0到n之间的随机int值，包含0而不包含n。
15 |
16 | public long nextLong()
17 | //生成一个随机的long值，该值介于long的区间，也就是-2^63到2^63-1之间。
18 |
19 | public void setSeed(long seed)
20 | //重新设置Random对象中的种子数。设置完种子数以后的Random对象和相同种子数使用new关键字
   | 创建出的Random对象相同。
```

6. Scanner类

`Scanner` 类提供了实用的从输入流 `InputStream` 读取格式数据的方法，很基础很有用。

输入流就是提供数据的数据源，标准输入流 `System.in` 指的就是键盘。

构造方法

```
1 Scanner(InputStream source)           //构造一个新的 Scanner ，产生从指定输入流读取
   的值。
2 Scanner(File source)                   //构造一个新的 Scanner ，产生从指定文件读取的
   值。
3 Scanner(String source)                  //构造一个新的 Scanner ，产生从指定字符串读取
   的值。
```

常用方法

`Scanner` 输入时默认分割符是空格。

当从输入流读入数据时，由于输入数据是源源不断的，而我们需要的数据可以看作是一段一段的，由 `delimiter` 进行划分，`Scanner` 默认分割符 `delimiter` 是空格，可以通过 `useDelimiter()` 设置分隔符。

```
1 void close()                           //关闭Scanner
2 boolean hasNext()                       //判断输入流当前位置后，是否还有下一段输
   入，读取停止时等待读取
3 boolean hasNext(String regex)           //如果输入流当前位置后下一字符串与正则
   表达式regex匹配，返回true
4 boolean hasNextBigInteger()             //判断是否有下一个BigInteger读入
5 boolean hasNextBoolean()
6 boolean hasNextByte()
7 boolean hasNextDouble()
8 boolean hasNextInt()
9 boolean hasNextLine()                   //判断输入流当前位置后，是否还有下一段输
   入，读取停止时跳出读取
10 boolean hasNextLong()
11 boolean hasNextShort()
12
13 String next()                           //返回读取的字符串
14 String nextLine()                       //返回读取的字符串
15 BigInteger nextBigInteger()             //返回下一个BigInteger
16 boolean nextBoolean()
17 byte nextByte()
18 double nextDouble()
19 float nextFloat()
20 int nextInt()
21 long nextLong()
22 short nextShort()
```

```
23
24 Scanner useDelimiter(String pattern)    //将此Scanner的分隔模式设置指定
    pattern
```

停止输入

有时我们会遇到从键盘（`System.in`）循环读入的情况，读入到某个截至标志或是直到输入流结束。这个时候我们会使用while循环读入：

```
1 Scanner cin=new Scanner("This is an InputStream!");
2 while(cin.hasNext())                //错误方式，无法跳出循环
3
4 //方式1
5 while(cin.hasNextLine()){
6     //...
7 }
8 //方式2
9 while(cin.hasNext("!")){
10     //...
11 }
```

当使用 `while(cin.hasNext())` 时到达输入流末尾后，它不会跳出循环而是继续等待输入，因此不能用。

方式1，会在到达输入流末尾后跳出循环。

方式2的 `hasNext(String regex)` 会在读入到与正则表达式 `pattern` 相匹配的是否返回 `true`，因此也可以跳出循环。关于正则表达式的内容，可以参考字符串部分里有关正则表达式的内容。

注意：当我们的输入源（输入流）是文件或是字符串的时候，`hasNext()`，`hasNextLine()` 都可以正确判别输入流的结束从而退出循环。

next与nextLine的区别

`next`与`nextLine`都可以获取一段字符串，二者区别如下：

`next`方法：

- 一定要读取到有效字符后才可以结束输入。
- 对输入有效字符之前遇到的空白，`next()` 方法会自动将其去掉。
- 只有输入有效字符后才将其后面输入的空白作为分隔符或者结束符。
- `next()` 不能得到带有空格的字符串。

```

1 public static void main(String[] args) {
2     Scanner cin=new Scanner(System.in);
3     while(!cin.hasNext("#")){
4         System.out.println(cin.next());
5     }
6 }
7 /*Input:This is a test #
8 Output:
9 This
10 is
11 a
12 test
13 */

```

nextLine方法

- 以Enter为结束符,也就是说 `nextLine()` 方法返回的是输入回车之前的所有字符;
- 字符串可以包含空格。

```

1 public static void main(String[] args) {
2     Scanner cin = new Scanner(System.in);
3     while (!cin.hasNext("!")) {
4         System.out.println(cin.nextLine());
5     }
6 }
7 /*Input:
8 first line    end
9 second line  end
10 !
11
12 Output:
13 first line    end
14 second line  end
15 */
16

```

设置分隔符, 当读取格式化数据是, `Scanner` 采用的默认分隔符 `delimiter` 是空格 (`nextLine`、`hasNextLine` 除外), 可以通过 `useDelimiter()` 设置分隔符。

```

1 public static void main(String[] args) {
2     Scanner cin = new Scanner(System.in);
3     cin.useDelimiter(",");
4     while(cin.hasNextInt()){
5         System.out.println(cin.nextInt());
6     }
7 }
8 /*Input: 102030
9 Output:
10 1
11 2
12 3
13 */

```

7. System.out

`System.out` 是一个静态输出流对象，我们可以通过它将字符串输出到控制台中。

常用方法

```

1 System.out.print(String str) //打印str
2 System.out.println(String str) //打印str并在末尾追加换行
3 System.out.printf(String format,Object ... args) //和C语言的printf函数一样，
  支持格式字符串

```

重定向

`Scanner` 可以传入一个 `File` 对象读取文件内容，我们还可以通过对 `System.out` 重定向使得可以向文件中输出内容。

```

1 System.setOut(PrintStream)

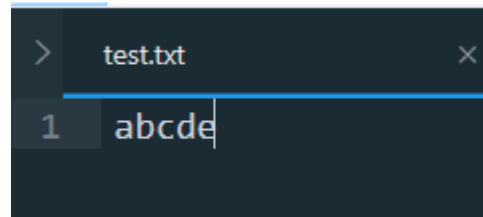
```

```

1 public static void main(String[] args) throws FileNotFoundException {
2     Scanner cin = new Scanner(System.in);
3     System.setOut(new PrintStream("./test.txt")); //写入根目录下的
  test.txt文件
4     while(!cin.hasNext("End")){
5         System.out.print(cin.next());
6     }
7 }
8 /*Output:
9 a b c d e
10 End
11 */

```

结果



```
> test.txt ×  
1 abcde
```