第 17 天常用 API

今日内容介绍

- 基本类型包装类
- System
- Math
- Arrays
- BigInteger
- BigDecimal

第 1 章　基本类型包装类

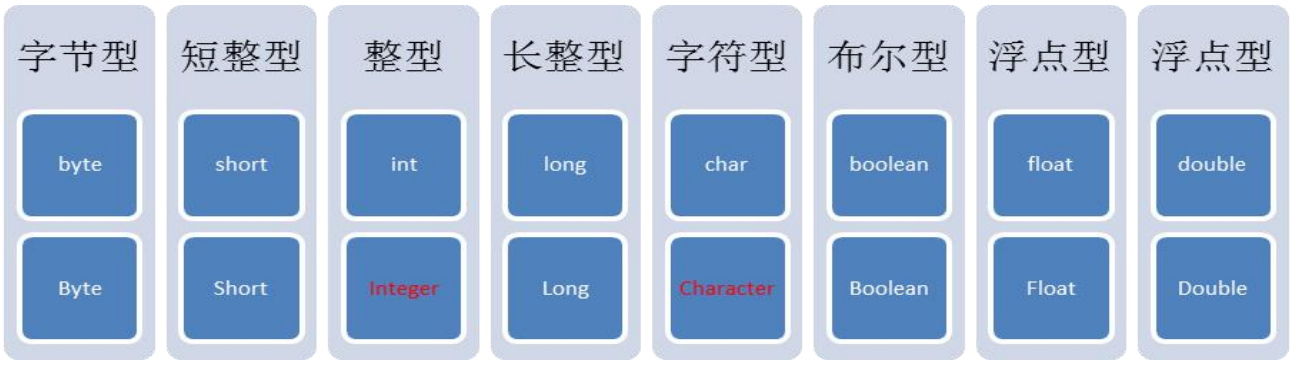
大家回想下 ,在第二天我们学习 Java 中的基本数据类型时 ,说 Java 中有 8 种基本的数据类型，可是这些数据是基本数据，想对其进行复杂操作，变的很难。怎么办呢？

1.1基本类型包装类概述

在实际程序使用中，程序界面上用户输入的数据都是以字符串类型进行存储的。而程序开发中，我们需要把字符串数据，根据需求转换成指定的基本数据类型，如年龄需要转换成 int 类型，考试成绩需要转换成 double 类型等。那么，想实现字符串与基本数据之间转换怎么办呢？

Java 中提供了相应的对象来解决该问题，基本数据类型对象包装类：java 将基本数据类型值封装成了对象。封装成对象有什么好处？可以提供更多的操作基本数值的功能。

8 种基本类型对应的包装类如下：



其中需要注意 int 对应的是 Integer，char 对应的 Character，其他 6 个都是基本类型首字母大写即可。

基本数据类型对象包装类特点：用于在基本数据和字符串之间进行转换。

- 将字符串转成基本类型：

static byte	**parseByte**(String s) 将 string 参数解析为有符号的十进制 byte。
static short	**parseShort**(String s) 将字符串参数解析为有符号的十进制 short。
static int	**parseInt**(String s) 将字符串参数作为有符号的十进制整数进行解析。
static long	**parseLong**(String s) 将 string 参数解析为有符号十进制 long。
static float	**parseFloat**(String s) 返回一个新的 float 值，该值被初始化为用指定 String 表示的值，
static double	**parseDouble**(String s) 返回一个新的 double 值，该值被初始化为用指定 String 表示的值，
static boolean	**parseBoolean**(String s) 将字符串参数解析为 boolean 值。

parseXXX(String s);其中 XXX 表示基本类型，参数为可以转成基本类型的字符串，如果字符串无法转成基本类型，将会发生数字转换的问题 NumberFormatException

```
System.out.println(Integer.parseInt("123") + 2);
//打印结果为 125
```

- 将基本数值转成字符串有 3 种方式：
  - 基本类型直接与""相连接即可；34+""

- 调用 String 的 valueOf 方法；**String.valueOf(34)** ；

static String	**valueOf**(boolean b) 返回 boolean 参数的字符串表示形式。
static String	**valueOf**(char c) 返回 char 参数的字符串表示形式。
static String	**valueOf**(double d) 返回 double 参数的字符串表示形式。
static String	**valueOf**(float f) 返回 float 参数的字符串表示形式。
static String	**valueOf**(int i) 返回 int 参数的字符串表示形式。
static String	**valueOf**(long l) 返回 long 参数的字符串表示形式。
static String	**valueOf**(Object obj) 返回 Object 参数的字符串表示形式。

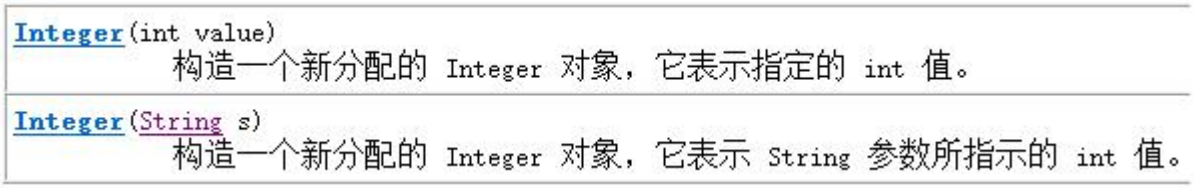
- 调用包装类中的 toString 方法；**Integer.toString(34)** ；

String	**toString**() 返回表示此 XXX 数据类型 值的 String 对象。
static String	**toString**(byte b) 返回表示指定 byte 的一个新 String 对象。
static String	**toString**(short s) 返回表示指定 short 的一个新 String 对象。
static String	**toString**(int i) 返回一个表示指定整数的 String 对象。
static String	**toString**(long i) 返回表示指定 long 的 String 对象。
static String	**toString**(float f) 返回 float 参数的字符串表示形式。
static String	**toString**(double d) 返回 double 参数的字符串表示形式。
static String	**toString**(char c) 返回一个表示指定 char 值的 String 对象。
static String	**toString**(boolean b) 返回一个表示指定布尔值的 String 对象。

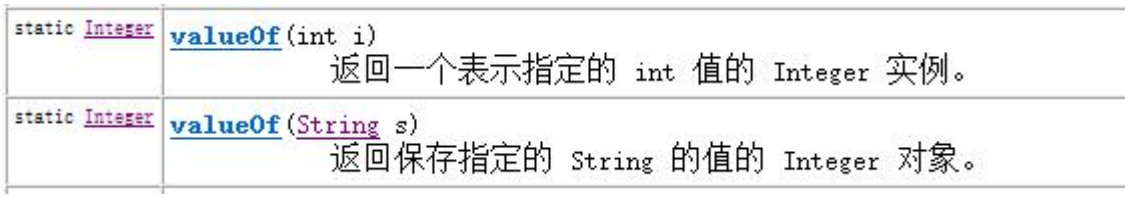
1.2基本类型和对象转换

使用 int 类型与 Integer 对象转换进行演示，其他基本类型转换方式相同。

- 基本数值---->包装对象




```
Integer i = new Integer(4);//使用构造函数函数
Integer ii = new Integer("4");//构造函数中可以传递一个数字字符串
```




```
Integer iii = Integer.valueOf(4);//使用包装类中的 valueOf 方法
Integer iiii = Integer.valueOf("4");//使用包装类中的 valueOf 方法
```

- 包装对象---->基本数值


```
int num = i.intValue();
```

1.3自动装箱拆箱

在需要的情况下，基本类型与包装类型可以通用。有些时候我们必须使用引用数据类型时，可以传入基本数据类型。

比如：

基本类型可以使用运算符直接进行计算，但是引用类型不可以。而基本类型包装类作为引用类型的一种却可以计算，原因在于，Java”偷偷地”自动地进行了对象向基本数据类型的转换。

相对应的，引用数据类型变量的值必须是 new 出来的内存空间地址值，而我们可以将一个基本

类型的值赋值给一个基本类型包装类的引用。原因同样在于 Java 又”偷偷地”自动地进行了基本数据类型向对象的转换。

- 自动拆箱：对象转成基本数值
- 自动装箱：基本数值转成对象

```
Integer i = 4;//自动装箱。相当于 Integer i = Integer.valueOf(4);
i = i + 5;//等号右边：将 i 对象转成基本数值(自动拆箱) i.intValue() + 5; 加法运算完成后，再次装箱，把基本数值转成对象。
```

- 自动装箱(byte 常量池)细节的演示

当数值在 byte 范围之内时，进行自动装箱，不会新创建对象空间而是使用医来已有的空间。

```
Integer a = new Integer(3);
Integer b = new Integer(3);
System.out.println(a==b);//false
System.out.println(a.equals(b));//true

System.out.println("----------------------");
Integer x = 127;
Integer y = 127;
//在 jdk1.5 自动装箱时，如果数值在 byte 范围之内，不会新创建对象空间而是使用原来已有的空间。
System.out.println(x==y); //true
System.out.println(x.equals(y)); //true
```

第 2 章　System 类

2.1概念

在 API 中 System 类介绍的比较简单，我们给出定义，System 中代表程序所在系统，提供了对应的一些系统属性信息，和系统操作。

System 类不能手动创建对象，因为构造方法被 private 修饰，阻止外界创建对象。System 类中的都是 static 方法，类名访问即可。在 JDK 中，有许多这样的类。

2.2常用方法

static long	currentTimeMillis() 返回以毫秒为单位的当前时间。
static void	exit(int status) 终止当前正在运行的 Java 虚拟机。
static void	gc() 运行垃圾回收器。
static String	getProperty(String key) 获取指定键指示的系统属性。

- currentTimeMillis() 获取当前系统时间与 1970 年 01 月 01 日 00:00 点之间的毫秒差值
- exit(int status) 用来结束正在运行的 Java 程序。参数传入一个数字即可。通常传入 0 记为正常状态，其他为异常状态
- gc() 用来运行 JVM 中的垃圾回收器，完成内存中垃圾的清除。
- getProperty(String key) 用来获取指定键(字符串名称)中所记录的系统属性信息

键	相关值的描述
java.version	Java 运行时环境版本
java.vendor	Java 运行时环境供应商
java.vendor.url	Java 供应商的 URL
java.home	Java 安装目录
java.vm.specification.version	Java 虚拟机规范版本
java.vm.specification.vendor	Java 虚拟机规范供应商
java.vm.specification.name	Java 虚拟机规范名称
java.vm.version	Java 虚拟机实现版本
java.vm.vendor	Java 虚拟机实现供应商
java.vm.name	Java 虚拟机实现名称
java.specification.version	Java 运行时环境规范版本
java.specification.vendor	Java 运行时环境规范供应商
java.specification.name	Java 运行时环境规范名称
java.class.version	Java 类格式版本号
java.class.path	Java 类路径
java.library.path	加载库时搜索的路径列表
java.io.tmpdir	默认的临时文件路径
java.compiler	要使用的 JIT 编译器的名称
java.ext.dirs	一个或多个扩展目录的路径
os.name	操作系统的名称
os.arch	操作系统的架构
os.version	操作系统的版本
file.separator	文件分隔符（在 UNIX 系统中是“/”）
path.separator	路径分隔符（在 UNIX 系统中是“:”）
line.separator	行分隔符（在 UNIX 系统中是“/n”）
user.name	用户的账户名称
user.home	用户的主目录
user.dir	用户的当前工作目录

- arraycopy 方法，用来实现将源数组部分元素复制到目标数组的指定位置

2.3System 类的方法练习

- 练习一：验证 for 循环打印数字 1-9999 所需要使用的时间（毫秒）

```
public static void main(String[] args) {
```

```
        long start = System.currentTimeMillis();
        for (int i=0; i<10000; i++) {
            System.out.println(i);
        }
        long end = System.currentTimeMillis();
        System.out.println("共耗时毫秒：" + (end-start) );
    }
```

- 练习二：将 src 数组中前 3 个元素，复制到 dest 数组的前 3 个位置上

  复制元素前：src 数组元素[1,2,3,4,5]，dest 数组元素[6,7,8,9,10]

  复制元素后：src 数组元素[1,2,3,4,5]，dest 数组元素[1,2,3,9,10]

```
public static void main(String[] args) {
    int[] src = new int[]{1,2,3,4,5};
    int[] dest = new int[]{6,7,8,9,10};
    System.arraycopy( src, 0, dest, 0, 3);
    代码运行后：两个数组中的元素发生了变化
    src 数组元素[1,2,3,4,5]
    dest 数组元素[1,2,3,9,10]
}
```

- 练习三：循环生成 100-999 之间的的三位数并进行打印该数，当该数能被 10 整除时，结束运行的程序

```
public static void main(String[] args){
    Random random = new Random();
    while(true){
        int number = random.nextInt(900)+100; //0-899 + 100
        if (nmumber % 10 == 0) {
            System.exit(0);
        }
    }
}
```

第 3 章　Math 类

3.1概念

Math 类是包含用于执行基本数学运算的方法的数学工具类，如初等指数、对数、平方根和三角函数。

类似这样的工具类，其所有方法均为静态方法，并且一般不会创建对象。如 System 类

3.2常用方法

static double	**abs**(double a) 返回 double 值的绝对值。
static double	**ceil**(double a) 返回最小的（最接近负无穷大）double 值，该值大于等于参数，并等于某个整数。
static double	**floor**(double a) 返回最大的（最接近正无穷大）double 值，该值小于等于参数，并等于某个整数。
static double	**max**(double a, double b) 返回两个 double 值中较大的一个。
static double	**min**(double a, double b) 返回两个 double 值中较小的一个。
static double	**pow**(double a, double b) 返回第一个参数的第二个参数次幂的值。
static double	**random**() 返回带正号的 double 值，该值大于等于 0.0 且小于 1.0。
static long	**round**(double a) 返回最接近参数的 long。

- abs 方法,结果都为正数

```
double d1 = Math.abs(-5); // d1 的值为 5
double d2 = Math.abs(5); // d2 的值为 5
```

- ceil 方法，结果为比参数值大的最小整数的 double 值

```
double d1 = Math.ceil(3.3); //d1 的值为 4.0
double d2 = Math.ceil(-3.3); //d2 的值为 -3.0
double d3 = Math.ceil(5.1); // d3 的值为 6.0
```

- floor 方法，结果为比参数值小的最大整数的 double 值

```
double d1 = Math.floor(3.3); //d1 的值为 3.0
double d2 = Math.floor(-3.3); //d2 的值为-4.0
double d3 = Math.floor(5.1); //d3 的值为 5.0
```

- max 方法，返回两个参数值中较大的值

```
double d1 = Math.max(3.3, 5.5); //d1 的值为 5.5
double d2 = Math.max(-3.3, -5.5); //d2 的值为-3.3
```

- min 方法，返回两个参数值中较小的值

```
double d1 = Math.min(3.3, 5.5); //d1 的值为 3.3
double d2 = Math.max(-3.3, -5.5); //d2 的值为-5.5
```

- pow 方法，返回第一个参数的第二个参数次幂的值

```
double d1 = Math.pow(2.0, 3.0); //d1 的值为 8.0
double d2 = Math.pow(3.0, 3.0); //d2 的值为 27.0
```

- round 方法，返回参数值四舍五入的结果

```
double d1 = Math.round(5.5); //d1 的值为 6.0
double d2 = Math.round(5.4); //d2 的值为 5.0
```

- random 方法，产生一个大于等于 0.0 且小于 1.0 的 double 小数

```
double d1 = Math.random();
```

第 4 章　Arrays 类

4.1概念

此类包含用来操作数组（比如排序和搜索）的各种方法。需要注意，如果指定数组引用为 null，则访问此类中的方法都会抛出空指针异常 NullPointerException。

4.2常用方法

static int	**binarySearch**(int[] a, int key) 使用二分搜索法来搜索指定的 int 型数组，以获得指定的值。
static void	**sort**(int[] a) 对指定的 int 型数组按数字升序进行排序。
static String	**toString**(int[] a) 返回指定数组内容的字符串表示形式。

- sort 方法，用来对指定数组中的元素进行排序（元素值从小到大进行排序）

```
//源 arr 数组元素{1,5,9,3,7}，进行排序后 arr 数组元素为{1,3,5,7,9}
```

```
int[] arr = {1,5,9,3,7};
Arrays.sort( arr );
```

- toString 方法，用来返回指定数组元素内容的字符串形式

```
int[] arr = {1,5,9,3,7};
String str = Arrays.toString(arr); // str 的值为[1, 3, 5, 7, 9]
```

- binarySearch 方法，在指定数组中，查找给定元素值出现的位置。若没有查询到，返回位置为-1。要求该数组必须是个有序的数组。

```
int[] arr = {1,3,4,5,6};
int index = Arrays.binarySearch(arr, 4); //index 的值为 2
int index2= Arrasy.binarySearch(arr, 2); //index2 的值为-1
```

4.3Arrays 类的方法练习

- 练习一：定义一个方法，接收一个数组，数组中存储 10 个学生考试分数，该方法要求返回考试分数最低的后三名考试分数。

```
public static int[] method(double[] arr){
    Arrays.sort(arr); //进行数组元素排序（元素值从小到大进行排序）
    int[] result = new int[3]; //存储后三名考试分数
    System.arraycopy(arr, 0, result, 0, 3);//把 arr 数组前 3 个元素复制到 result 数组中
    return result;
}
```

第 5 章　大数据运算

5.1BigInteger

java 中 long 型为最大整数类型,对于超过 long 型的数据如何去表示呢.在 Java 的世界中,超过 long 型的整数已经不能被称为整数了,它们被封装成 BigInteger 对象.在 BigInteger 类中,实现四则运算都是方法来实现,并不是采用运算符.

BigInteger 类的构造方法:

构造方法摘要

BigInteger(byte[] val) 将包含 BigInteger 的二进制补码表示形式的 byte 数组转换为 BigInteger。
BigInteger(int signum, byte[] magnitude) 将 BigInteger 的符号-数量表示形式转换为 BigInteger。
BigInteger(int bitLength, int certainty, Random rnd) 构造一个随机生成的正 BigInteger，它可能是一个具有指定 bitLength 的素数。
BigInteger(int numBits, Random rnd) 构造一个随机生成的 BigInteger，它是在 0 到 $(2^{numBits} - 1)$（包括）范围内均匀分布的值。
BigInteger(String val) 将 BigInteger 的十进制字符串表示形式转换为 BigInteger。
BigInteger(String val, int radix) 将指定基数的 BigInteger 的字符串表示形式转换为 BigInteger。

构造方法中,采用字符串的形式给出整数

四则运算代码：

```
/
public static void main(String[] args) {
        //大数据封装为 BigInteger 对象
         BigInteger big1 = new BigInteger("12345678909876543210");
         BigInteger big2 = new BigInteger("98765432101234567890");
         //add 实现加法运算
         BigInteger bigAdd = big1.add(big2);
         //subtract 实现减法运算
         BigInteger bigSub = big1.subtract(big2);
         //multiply 实现乘法运算
         BigInteger bigMul = big1.multiply(big2);
         //divide 实现除法运算
         BigInteger bigDiv = big2.divide(big1);
}
```

5.2BigDecimal

在程序中执行下列代码,会出现什么问题?

```
System.out.println(0.09 + 0.01);
System.out.println(1.0 - 0.32);
System.out.println(1.015 * 100);
System.out.println(1.301 / 100);
```

double 和 float 类型在运算中很容易丢失精度,造成数据的不准确性,Java 提供我们 BigDecimal

类可以实现浮点数据的高精度运算

构造方法如下:

构造方法摘要

BigDecimal(BigInteger val) 将 BigInteger 转换为 BigDecimal。
BigDecimal(BigInteger unscaledVal, int scale) 将 BigInteger 非标度值和 int 标度转换为 BigDecimal。
BigDecimal(BigInteger unscaledVal, int scale, MathContext mc) 将 BigInteger 非标度值和 int 标度转换为 BigDecimal（根据上下文设置进行舍入）。
BigDecimal(BigInteger val, MathContext mc) 将 BigInteger 转换为 BigDecimal（根据上下文设置进行舍入）。
BigDecimal(char[] in) 将 BigDecimal 的字符数组表示形式转换为 BigDecimal，接受与 BigDecimal(String) 构造方法相同的字符序列。
BigDecimal(char[] in, int offset, int len) 将 BigDecimal 的字符数组表示形式转换为 BigDecimal，接受与 BigDecimal(String) 构造方法相同的字符序列，同时允许指定子数组。
BigDecimal(char[] in, int offset, int len, MathContext mc) 将 BigDecimal 的字符数组表示形式转换为 BigDecimal，接受与 BigDecimal(String) 构造方法相同的字符序列，同时允许指定子数组，并根据上下文设置进行舍入。
BigDecimal(char[] in, MathContext mc) 将 BigDecimal 的字符数组表示形式转换为 BigDecimal，接受与 BigDecimal(String) 构造方法相同的字符序列（根据上下文设置进行舍入）。
BigDecimal(double val) 将 double 转换为 BigDecimal，后者是 double 的二进制浮点值准确的十进制表示形式。
BigDecimal(double val, MathContext mc) 将 double 转换为 BigDecimal（根据上下文设置进行舍入）。
BigDecimal(int val) 将 int 转换为 BigDecimal。
BigDecimal(int val, MathContext mc) 将 int 转换为 BigDecimal（根据上下文设置进行舍入）。
BigDecimal(long val) 将 long 转换为 BigDecimal。
BigDecimal(long val, MathContext mc) 将 long 转换为 BigDecimal（根据上下文设置进行舍入）。
BigDecimal(String val) 将 BigDecimal 的字符串表示形式转换为 BigDecimal。
BigDecimal(String val, MathContext mc) 将 BigDecimal 的字符串表示形式转换为 BigDecimal，接受与 BigDecimal(String) 构造方法相同的字符串（按照上下文设置进行舍入）。

建议浮点数据以字符串形式给出,因为参数结果是可以预知的

实现加法减法乘法代码如下:

```
public static void main(String[] args) {
        //大数据封装为 BigDecimal 对象
        BigDecimal big1 = new BigDecimal("0.09");
        BigDecimal big2 = new BigDecimal("0.01");
        //add 实现加法运算
        BigDecimal bigAdd = big1.add(big2);

        BigDecimal big3 = new BigDecimal("1.0");
        BigDecimal big4 = new BigDecimal("0.32");
        //subtract 实现减法运算
        BigDecimal bigSub = big3.subtract(big4);

        BigDecimal big5 = new BigDecimal("1.105");
        BigDecimal big6 = new BigDecimal("100");
        //multiply 实现乘法运算
        BigDecimal bigMul = big5.multiply(big6);
```

对于浮点数据的除法运算,和整数不同,可能出现无限不循环小数,因此需要对所需要的位数进行保留和选择舍入模式

BigDecimal	**divide**(BigDecimal divisor, int scale, int roundingMode) 返回一个 BigDecimal，其值为 (this / divisor)，其标度为指定标度。

字段摘要	
static BigDecimal	**ONE** 值为 1，标度为 0。
static int	**ROUND_CEILING** 接近正无穷大的舍入模式。
static int	**ROUND_DOWN** 接近零的舍入模式。
static int	**ROUND_FLOOR** 接近负无穷大的舍入模式。
static int	**ROUND_HALF_DOWN** 向“最接近的”数字舍入，如果与两个相邻数字的距离相等，则为上舍入的舍入模式。
static int	**ROUND_HALF_EVEN** 向“最接近的”数字舍入，如果与两个相邻数字的距离相等，则向相邻的偶数舍入。
static int	**ROUND_HALF_UP** 向“最接近的”数字舍入，如果与两个相邻数字的距离相等，则为向上舍入的舍入模式。
static int	**ROUND_UNNECESSARY** 断言请求的操作具有精确的结果，因此不需要舍入。
static int	**ROUND_UP** 舍入远离零的舍入模式。

第 6 章　总结

6.1知识点总结

- 基本类型包装类
  - 8 种基本类型对应的包装类

基本类型	包装类
byte	Byte
short	Short
int	Integer
log	Long
float	Float
double	Double
char	Character

boolean Boolean

- 自动装箱、自动拆箱
  - 自动装箱：基本数值转成对象（int → Integer）
  - 自动拆箱：对象转成基本数值（Integer → int）
- 常用方法

  public int parseInt(String str):把字符串转成基本类型 int

  public static String toString(int x):把基本类型 int 转成字符串

  public static Integer valueOf(int x):把基本类型 i 字符串转成 Integer 对象

  public int intValue():以 int 类型返回该包装类对象的值

- System 类: 系统属性信息工具类
  - public static long currentTimeMillis()：获取当前系统时间与 1970 年 01 月 01 日 00:00 点之间的毫秒差值
  - public static void exit(int status)：用来结束正在运行的 Java 程序。参数传入一个数字即可。通常传入 0 记为正常状态，其他为异常状态
  - public static void gc()：用来运行 JVM 中的垃圾回收器，完成内存中垃圾的清除。
  - public static String getProperties()：用来获取指系统属性信息
- Arrays 类：数组操作工具类
  - public static void sort 方法，用来对指定数组中的元素进行排序（元素值从小到大进行排序）
  - public static String toString 方法，用来返回指定数组元素内容的字符串形式
  - public static void binarySearch 方法，在指定数组中，查找给定元素值出现的位置。若没

有查询到，返回位置为-插入点-1。要求该数组必须是个有序的数组

- Math 类：数学运算工具类
  - abs 方法,结果都为正数
  - ceil 方法，结果为比参数值大的最小整数的 double 值
  - floor 方法，结果为比参数值小的最大整数的 double 值
  - max 方法，返回两个参数值中较大的值
  - min 方法，返回两个参数值中较小的值
  - pow 方法，返回第一个参数的第二个参数次幂的值
  - round 方法，返回参数值四舍五入的结果
  - random 方法，产生一个大于等于 0.0 且小于 1.0 的 double 小数