Java中的数据类型与八种基本数据类型对象包装类

1. **数据类型：**
2. JAVA语言是一种**强类型语言**，对于每一种数据都定义了明确的具体数据类型，在内存中分配了**不同大小**的内存空间。
3. Java的数据类型：
4. **原生数据类型**（Primitive Data Type）：共8种。
5. **三种引用型类型（对象类型）**（Reference Type）：数组类型，类类型（利用类定义的引用变量）、接口型引用。引用型变量要么指向某个对象，要么就是**null**。注意：**null**是一个常量。



**整数类型默认情况下都是int类型，浮点类型数据默认情况下都是double类型。**

1. 八种原生数据类型及包装类：
2. 整型：**int**；🡺**Integer**
3. 字节型：**byte** 🡺**Byte**（与整型的差别只在范围比较小，是-128—127范围内的256个整数）；
4. 短整型（16位）**short**🡺**Short**；
5. 长整型（64位）**long**🡺**Long**；
6. 单精度浮点型**float**🡺**Float**
7. 双精度浮点型**double**🡺**Double**
8. 字符型**char**🡺**Character**：
9. 布尔类型**boolean**🡺**Boolean**：值只有两种true 和false。
10. JAVA中的八种原生数据类型不是**对象**，必要时可以转换成相应**包装类对象**。
11. Java中除了**八种原生数据类型**之外，其他的都是**类，且**八种基本数据类型也都有自己**对应的包装类**。
12. 类类型的引用可以指向本类及其子类的对象；接口类型的引用可以用于指向实现接口的子类的对象。这就是**多态**的体现。
13. Java使用的是Unicode编码，Unicode编码中一个字符对应两个字节，故java中**char类型**数据占据两个字节。
14. 基本数据类型的**取值范围**及在内存中的占用的**字节个数**：
15. **整数类型：**byte short int long  这四个类型都属于整数类型；**字符类型**也可以算作**整型数据**。
16. byte占1个字节，范围-128 – 127；
17. short 占2个字节；
18. char占2个字节；
19. int占**4个**字节；
20. long占**8个**字节。

默认情况下都把整数默认为int类型。int类型变量可以接收short和byte类型数据，short类型和byte类型数据会自动转成int类型。

1. **浮点类型：**
2. float占**4个**字节，
3. double占**8个**字节；

小数默认情况下都是**double型**的，所以在定义一个float类型小数时，需要加上一个f声明如 float a = 3.14f； 定义double类型就不需要注明，double a = 3.14；，以精确度高优先。

注意: **float类型数据的定义**：

float f0 = 99.0; //这样是错误的，因为默认小数位double型，必须进行**显式声明或显式类型转换**

float f2 = 99.0f;//显式声明为float型

float f1 = (float)99.0;//强制类型转换为float

1. 利用包装类获取**取值范围和所占字节数**：

System.out.println("byte : "+Byte.MIN\_VALUE+" ~ "+Byte.MAX\_VALUE+" 字节数 ："+Byte.BYTES); **// byte : -128 ~ 127 字节数 ：1**

System.out.println(" char 字节数 ："+Character.BYTES);**// char 字节数 ：2**

System.out.println("short : "+Short.MIN\_VALUE+" ~ "+Short.MAX\_VALUE+" 字节数 ："+Short.BYTES);**// short : -32768 ~ 32767 字节数 ：2**

System.out.println("int : "+Integer.MIN\_VALUE+" ~ "+Integer.MAX\_VALUE+" 字节数 ："+Integer.BYTES);**// int : -2147483648 ~ 2147483647 字节数 ：4**

System.out.println("float : "+Float.MIN\_VALUE+" ~ "+Float.MAX\_VALUE+" 字节数 ："+Float.BYTES);**// float : 1.4E-45 ~ 3.4028235E38 字节数 ：4**

System.out.println("long : "+Long.MIN\_VALUE+" ~ "+Long.MAX\_VALUE+" 字节数 ："+Long.BYTES);**// long : -9223372036854775808 ~ 9223372036854775807 字节数 ：8**

System.out.println("double : "+Double.MIN\_VALUE+" ~ "+Double.MAX\_VALUE+" 字节数 ："+Double.BYTES);**// double : 4.9E-324 ~ 1.7976931348623157E308 字节数 ：8**

1. **类型转换:**
2. **自动类型转换**：也叫隐式类型转换。

**表达式的数据类型自动提升：**

1. 在表达式中，所有的**byte类型、short类型、char类型**的值将被提升为int型；
2. 在表达式中，如果一个操作数为**long类型**，结果就为long类型；
3. 在表达式中，如果一个操作数为**float类型**，结果就为float类型；
4. 在表达式中，如果一个操作数为**double类型**，结果就为double类型。

实例：

System.out.println('a');//输出 a

System.out.println('a'+0);//输出 97 **隐式类型转换**

System.out.println((char)('a'+3));//输出 d **显式类型转换**

由于字符a并不是在表达式中，故以**原来的数据类型**显示，一旦byte、short、char在表达式中，就会先转变成int类型在进行计算。

1. **强制类型转换**：也叫**显式类型转换**。

System.out.println((char)('a'+3));//输出 d **显式类型转换**

1. **考点：**如果有三个变量，有其中两个变量相加，赋值给第三个变量，必须左边的变量的范围能够包含左边的变量的最大值的之和， 否则编译失败。

**例如： byte a = 3；**

**byte a = 4 + 5；//正确，因为右边是常量，判断没有超出字节的范围，所以合法**

**byte b = 4；**

**byte c = 5；**

**a = b + c；//报错，因为右边是变量，编译时不判断变量的大小值，只会提示可能超出字节范围，所以出错。 解决办法，要么对右边强制类型转换为byte，或者直接利用一个int变量接收即可。**

**int a = 3；**

**int b = 4；**

**int c = 5；**

**a = b + c；// 合法，为什么？因为int是默认的整数类型，如果两个int变量相加超过int范围，会有自动转换机制，即把超出范围舍弃，保留低位，但是会出现正负不正确问题。**

**此种情况，int类型、long类型、float类型 、 double类型都可以通过，是合法的。但是对于byte类型、short类型都是不合法的 。**

1. 考点：对于Java中的参数传递来说，原生类型数据的传递传递的是值，引用类型数据类型本质是传递的是地址。
2. **八种基本数据类型对象包装类**
3. 为了方便操作基本数据类型值，将其封装成了**对象**，在对象中定义了**属性和方法**，方便了和丰富了对此数据的操作。用于描述该对象的类称为**基本数据类型对象包装类**。
4. 八种原生数据类型及包装类：

（1） 整型：int 🡺 **Integer**

（2） 字节型：byte 🡺**Byte**

（3） 短整型（16位）short 🡺 **Short**；

（4） 长整型（64位）long 🡺**Long**；

（5） 单精度浮点型float 🡺 **Float**；

（6） 双精度浮点型double 🡺 **Double**；

（7） 字符型char 🡺 **Character**；

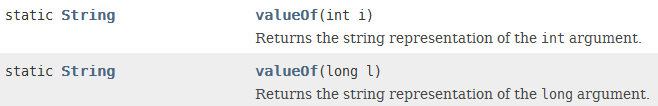
（8） 布尔类型boolean 🡺 **Boolean**。

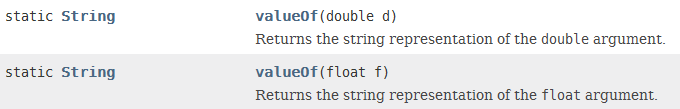
八种类型中只有int和char类名不一致，其他的都是将首字符大写即可。

**int** 对应类名为 **Integer**；**char** 对应类名为**Character** 。

1. 这些包装类都是**final修饰**的，不能被继承，当然其中的方法也不能被覆写的。这些类中的大部分方法都是**静态方法，可以利用类名进行调用，如parseXxx方法**，也有一部分方法是特有方法。
2. **valueOf方法**：注意： String类中**valueOf方法**都是静态方法：把**基本数据类型**转换成**字符串**。

其中八种基本数据类型都可以利用valueOf方法转换成字符串，只不过byte和short没有定义，但是可以利用**int**进行接收，隐式的类型转换。

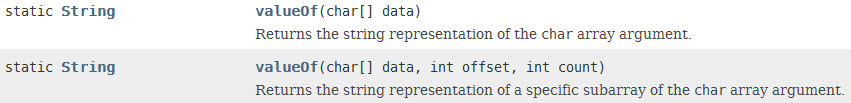








此外，对于字符数组也可以。

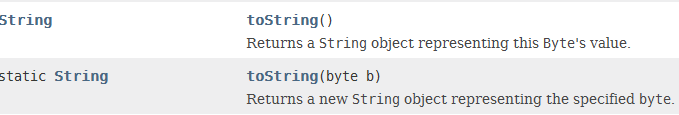




1. 这些包装类的一个主要作用就是**字符串与基本数据类型的转换**。
2. 这些包装类中重要的方法主要有：**parseXxx（String）方法、toString() 方法、valueOf（）方法等**。
3. **字符串和基本数据类型之间的转换方法**：
4. **字符串🡺基本数据类型：**
5. 方法1：利用**包装类的静态方法parseXxx（String str）方法**：参数为字符串，返回值为相应的基本数据类型值。

只有Character没有parseXxx方法。

1. 方法2：利用包装类的**构造方法，先把字符串封装成包装类对象，然后利用非静态方法xxxValue（）获取基本数据值**：
2. 方法3：利用**包装类的valueOf静态方法，先把字符串转换成包装类对象，其次利用xxxValue（）非静态方法获取基本数据类型的值**。
3. **基本数据类型🡺字符串：**
4. **方法1：基本数据类型 + “”**
5. **方法2：**利用**String类的静态方法valueOf（基本数据类型值），返回字符串。**
6. **方法:3：利用相应包装类的toString（）方法。有两个toString：其中一个是带有参数的静态方法，另外一个是不带的参数的成员方法。**

如：

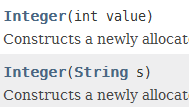
**既有静态的toString（基本数据类型）又有空参数的非静态方法toString（）方法。对于Integer和Long来说，还提供了转换成2、8、16进制及指定radix进制的toString静态方法。对于Float和Double来说，提供了转换成16进制的静态方法。**

**注意：String类和数据包装类都具有valueOf方法，其中String类是把数据转换成字符串，而数据包装类是把字符串转换成包装类对象。**

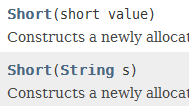
1. 包装类的**构造方法**介绍：

这些包装类除了**Character**之外，都有一个**参数为String**的构造方法，从而封装成对象相应的包装类对象。此外，这些包装类一般只有两个构造方法，一个是把**自己的基本数据类型**封装成**包装类对象**，一个就是把**字符串形式的值**封装成**包装类对象**。

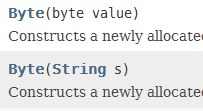
* 1. **Integer：**



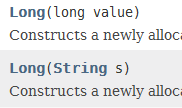
* 1. **Short：**



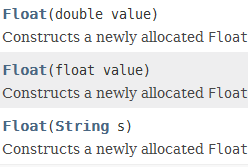
* 1. **Byte：**



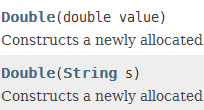
* 1. **Long：**



* 1. **Float**



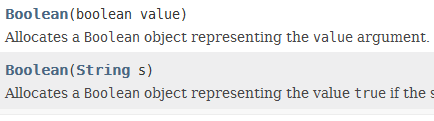
* 1. **Double**



* 1. **Character:**



* 1. **Boolean：**



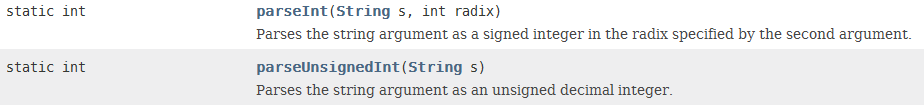
1. **包装类的静态方法parseXxx（String str）方法：只有Character没有parseXxx方法。**
2. Integer：



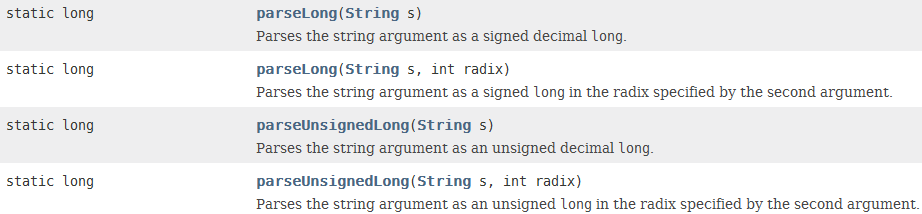
把字符串表示的数据当做指定进制进行转换：



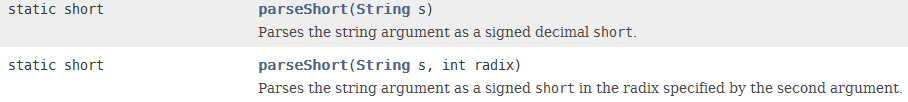
还有两个相应的**无符号数**操作：



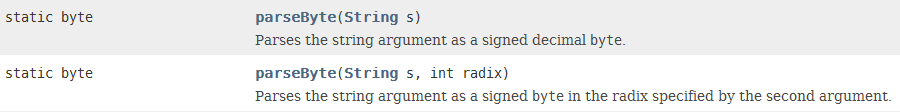
1. Long与Integer一样：



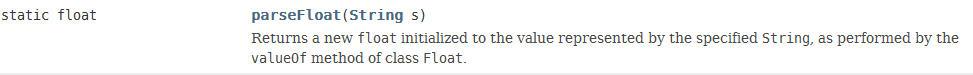
1. Short：



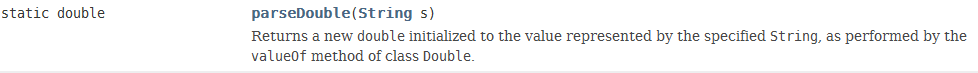
1. Byte:



1. Float:



1. Double:



1. Boolean：只有这一个静态方法。



1. **Character**:没有parse方法，因为没有意义，本来就是字符。

如：int i1 = Integer.parseInt("3335");

long l1 = Long.parseLong("4565346");

double d1 = Double.parseDouble("3535235.234534534");

boolean b1 = Boolean.parseBoolean("true");

1. 包装类的**toString（）静态方法**：

**既有静态的toString（基本数据类型）又有空参数的非静态方法toString（）方法。对于Integer和Long来说，还提供了转换成2、8、16进制及指定radix进制的toString静态方法。对于Float和Double来说，提供了转换成16进制的静态方法。**

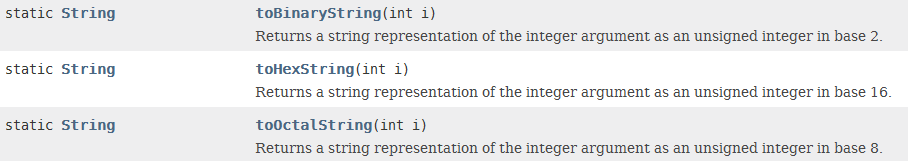
* 1. Integer：

**静态方法**： 可以指定**几进制，从而转换成相应进制的字符串**。

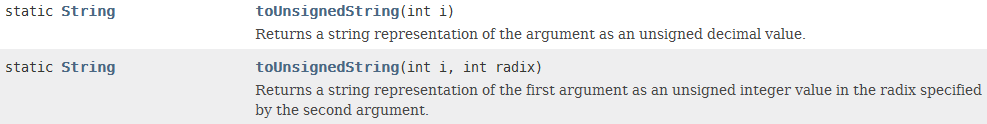




此外，已经定义了三个常用的进制，**2进制、8进制、16进制**等三个静态方法：



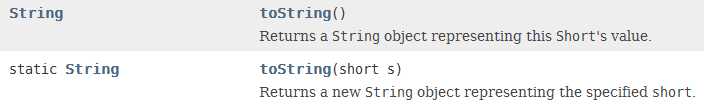
还有两个可以转成**无符号**的字符串形式：



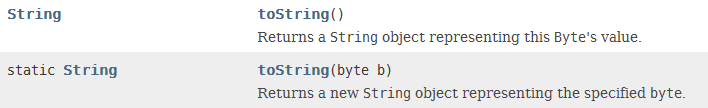
**非静态方法**：



* 1. Short：就这两个，一个静态的，一个非静态的。



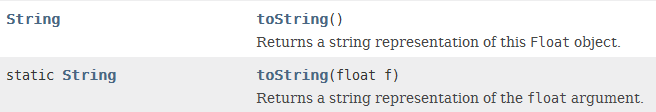
* 1. Byte：就这两个，一个静态的，一个非静态的。



* 1. Long：同Integer一样。

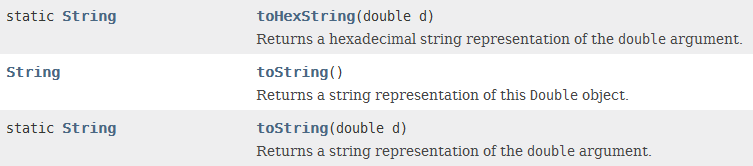


* 1. Float：多了一个toHexString（）.

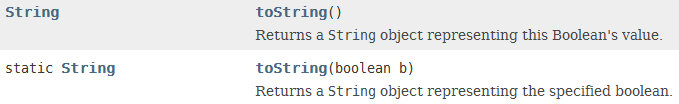




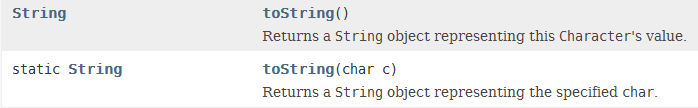
* 1. Double：同Float。



* 1. Boolean:



* 1. Character:



1. **包装类的valueOf（）静态方法**：

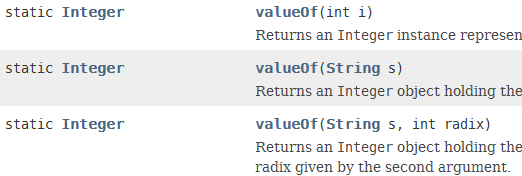
类似于包装类的**构造方法**的功能，返回**相应的包装类对象**。

**Integer、Byte、Short、Long**一样都有三个valueOf方法，都是静态的，返回都是包装类对象，第一个就是把基本数据类型值封装成对象，第二个是把字符串转换成包装类对象，第三个是指定字符串为特定的进制数，然后转换成包装类对象。

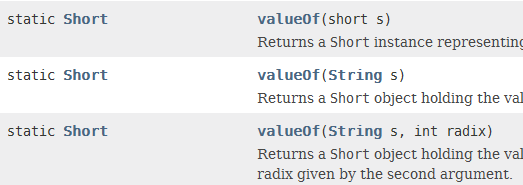
**Float、Double、Boolean** 都是只有前两个，没有指定的进制数。

**Character**只有一个valueOf方法，只是把字符封装成对象即可。

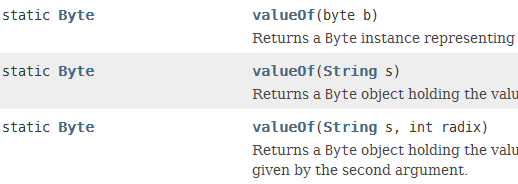
* + 1. Integer：



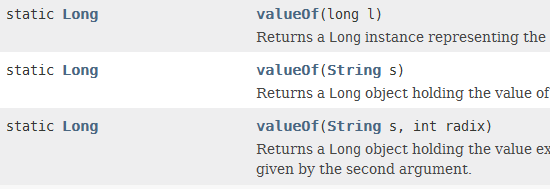
* + 1. Short :



* + 1. Byte：



* + 1. Long：



* + 1. Float ：



* + 1. Double：



* + 1. Boolean：

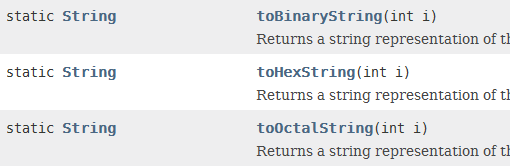


* + 1. Character：



1. 包装类的**equals方法**和**比较方法compareTo**：都是比较的值，因为都对Object的方法equals方法进行了**覆写**。
2. 从包装类对象获取此**对象的基本数据类型值**：利用**包装类的非静态方法xxxValue（）**。
3. Integer：intValue（）方法，
4. Short : shortValue（）方法；
5. Byte：byteValue（）方法；
6. Long：longValue（）方法；
7. Float ： floatValue（）方法；
8. Double：doubleValue（）方法；
9. Boolean：booleanValue（）方法。
10. Character：charValue（）方法。
11. 整数**不同进制**之间的转换：
12. Integer： toString方法。
    1. 以特定进制把**基本数据**转换成**字符串**；





例子：System.out.println(Integer.toBinaryString(255));

System.out.println(Integer.toHexString(255));

System.out.println(Integer.toOctalString(255));

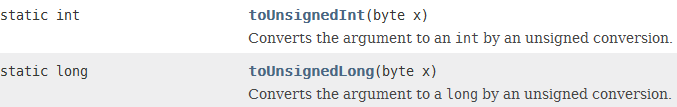
System.out.println(Integer.toString(266,25));//25进制，结果为af。

* 1. 指定**字符串为特定进制**转换成**十进制的基本数据或对象**

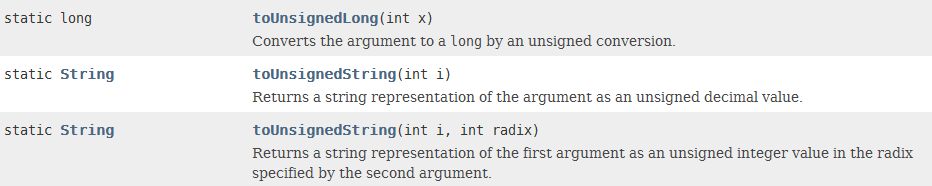




1. Short、Long、Byte都与Integer类似，不再赘述。
2. **将byte类型的数据转换成无符号的int类型，利用Byte的静态方法int toUnsignedInt（byte b）方法。类似可以转换为long类型。**



**类似的，int类型的数据转换成无符号的long类型，利用Integer的静态方法long toUnsignedLong(int x)。有两个方法直接转换成无符号的字符串。**



1. **基本数据类型的装箱和拆箱：**

**八个包装类都具有自动装箱和拆箱的功能。**

1. **当把基本数据类型值赋给引用（对应包装类引用或者父类引用）时，可以自动装箱；如 Integer i = 8；//自动装箱**
2. **当把基本数据对象与基本数据类型进行直接运算时，可以直接拆箱； 如：Integer ii = new Integer(8); int x = ii + 99;//可以自动拆箱**

**说明：JDK1.4之前，定义一个Integer对象，必须 Integer i = new Integer（99）；**

**自从JDK1.5 之后，只需要 Integer i = 99；即可。 JDK1.5之后，引入了自动装箱和自动拆箱。 i = i +55； 这叫做自动拆箱。但是引入了问题，就是i 有可能为null，所以需要健壮性的判断。**

1. **练习题：**
2. 单个字节创建对象的共享问题（**面试时有可能会考到**）：

Integer i1 = 127;

Integer i2 = 127;

System.out.println(i11 == i22);**//true**

Integer i3 = 128;

Integer i4 = 128;

System.out.println(i111 == i222**);//false**

Integer i1 = new Integer(127);

Integer i2 = new Integer(127);

System.out.println(i1 == i2);**//false**

* 1. **通过new创建的对象，肯定是不同的，一个new肯定对应一个新的对象；**
  2. **通过简化方式创建对象，即不利用new的方式：如** Integer i2 = 127;

**JDK1.5 之后，自动装箱时，如果装箱的是一个字节，则该数据会被共享，不会再次开辟空间，而是返回同一个对象。但是如果数值超出一个字节的范围-128--127，那么就会重新建立一个对象。**

1. **对一个字符串中的数值进行从小到大的排序：**

**字符串：45 56 2 33 7777 99 43234 454 -234 -1 3 -958 0**

**分析思路：**

* + 1. **对字符串进行切割，得到字符串数组：利用String的spilt方法；注意：字符串中的数值是用空格分隔开（不一定是一个），故regex = “ ”+”+”；**
    2. **把字符串数组转换成int数组：每个字符串转成int值利用Xxx类的parseXxx方法；**
    3. **利用Array的sort方法对int数组进行排序；**
    4. **然后把int数组再转成字符串，利用StringBuilder类的append方法。**

**代码如下：**

**package test.test1;**

**import java.util.Arrays;**

**public class SplitTransformToNumber {**

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

**String str = " 88 45 -22222 56 2 33 7777 99 43234 454 -234 -1 3 -958 0 ";**

**System.out.println("排序前 ： "+str+" 字符个数 ：\t "+str.length());**

**str = str.trim();//至关重要，如果不是把前后端的空格去掉，导致错误**

**String regex = " "+"+";**

**String[] strarr = str.split(regex);**

**int len = strarr.length;**

**int[] intarr = new int[len];**

**for (int i = 0; i < len; i++) {**

**intarr[i] = Integer.parseInt(strarr[i]);**

**}**

**Arrays.sort(intarr);**

**StringBuilder sb = new StringBuilder();**

**for (int i = 0; i < len; i++) {**

**if(i < len-1)**

**sb.append(intarr[i]+" ");**

**else**

**sb.append(intarr[i]);**

**}**

**System.out.println("排序后 ：\t "+sb+" 字符个数 :"+sb.length()+" 数字个数 ：\t "+len);**

**}**

**}**

**结果如下：**

