【济南中心】JAVA 编程阶梯：基础篇之第十三章

StringBuffer 类的概述：

StringBuffer 是字符串变量，它的对象是可以扩充和修改的。

它是线程安全的可变字符序列

StringBuffer 和 String 的区别：

String 是一个不可变的字符序列

StringBuffer 是一个可变的字符序列

StringBuffer 类的构造方法：

```
public final class StringBuffer extends AbstractStringBuilder implements
                Appendable, Serializable, CharSequence {
                                        public StringBuffer(){}//无参构造方法
                                        public StringBuffer(int capacity){
                                                        //指定容量的字符串缓冲区对象
                                                        super(capacity);
                                        }

                                        public StringBuffer(String str){
                                                        //指定字符串内容的字符串缓冲
区对象

                                                        super(str)
                                        }
                        }
```

Stringbuffer 方法的使用：

public int capacity()：返回当前容量。 理论值(不掌握)

public int length():返回长度（字符数）。 实际值

例子：

```
StringBuffer stringBuffer = new StringBuffer();//创建一个 StringBuffer 对象
stringBuffer.length();//获取 StringBuffer 的长度
```

StringBuffer 的添加功能：

* public StringBuffer append(String str):

* 可以把任意类型数据添加到字符串缓冲区里面,并返回字符串缓冲区本身

* public StringBuffer insert(int offset,String str):

* 在指定位置把任意类型的数据插入到字符串缓冲区里面,并返回字符串缓冲区本身

StringBuffer 的删除功能：

public StringBuffer deleteCharAt(int index)://删除指定位置的字符，并返回本身

public StringBuffer delete(int start,int end)://删除从指定位置开始指定位置结束的内容，并返回本身

StringBuffer 的替换和反转功能：

A:StringBuffer 的替换功能

public StringBuffer replace(int start,int end,String str)://从 start 开始到 end 用 str 替换

B:StringBuffer 的反转功能

public StringBuffer reverse()://字符串反转

StringBuffer 的截取功能及注意事项：

A:StringBuffer 的截取功能

public String substring(int start)://从指定位置截取到末尾

public String substring(int start,int end)://截取从指定位置开始到结束位置，包括开始位置，不包括结束位置

B:注意事项

注意:返回值类型不再是 StringBuffer 本身

StringBuffer 和 String 的相互转换：

A:String -- StringBuffer

* a:通过构造方法

* b:通过 append()方法

B:StringBuffer -- String

* a:通过构造方法

* b:通过 toString()方法

* c:通过 subString(0,length);

把数组转成字符串：

案例演示

* 需求：把数组中的数据按照指定个格式拼接成一个字符串

```
public static void main(String[] args){
String[] strings = { "a", "b", "c", "d" };
StringBuffer stringBuffer = new StringBuffer();
for (String string : strings){
stringBuffer.append(string);
}
System.out.println(stringBuffer.toString());
}
```

StringBuffer 和 StringBuilder 的区别：

StringBuffer 字符串变量（线程安全）

StringBuilder 字符串变量（非线程安全）

java.lang.StringBuilder 一个可变的字符序列是 5.0 新增的。此类提供一个与 StringBuffer 兼容的 API，但不保证同步。该类被设计用作 StringBuffer 的一个简易替换，用在字符串缓冲区被单个线程使用的时候（这种情况很普遍）。如果可能，建议优先采用该类，因为在大多数实现中，它比 StringBuffer 要快。两者的方法基本相同

冒泡排序：

```
public class bubbleSort {
  public   bubbleSort(){
            int
a[]={49,38,65,97,76,13,27,49,78,34,12,64,5,4,62,99,98,54,56,17,18,23,34,15,35,25,53,51};
          int temp=0;
          for(int i=0;i<a.length-1;i++){
                  for(int j=0;j<a.length-1-i;j++){
                  if(a[j]>a[j+1]){
                          temp=a[j];
                          a[j]=a[j+1];
                          a[j+1]=temp;
                  }
                  }
          }
          for(int i=0;i<a.length;i++)
                  System.out.println(a[i]);
  }
  }
```

选择排序：

```
public void SelectSort(int[] array) {
                int i, j, k;// 分别为有序区，无序区，无序区最小元素指针
                for (i = 0; i < array.length; i++) {
                        k = i;
                        for (j = i + 1; j < array.length; j++) {
                                if (array[j] < array[k])
                                        k = j;
                        }
                        if (k != i)// 若发现最小元素，则移动到有序区
                        {
                                int temp = array[k];
```

```
                        array[k] = array;
                        array = array[temp];
                    }
                }
            }
```

二分查找：

```
public static int branchSearch(int[] array, int searchNum) {
            if (array == null)
                    throw new NullPointerException("Null Referrence");
            if (array.length == 0)
                    throw new IllegalArgumentException("Array Length is Zero");
            int low = 0, high = array.length;
            int middle = (high + low) / 2;
            int index = -1;
            if (searchNum < array[0] || searchNum > array[array.length - 1])
                    return index;
            while (middle >= 0) {
                    if (array[middle] == searchNum) {
                            index = middle;
                            break;
                    }
                    if (searchNum > array[middle]) {
                            low = middle;
                    } else {
                            high = middle;
                    }
                    middle = (low + high) / 2;
            }

            return index;
    }
```

Arrays 类的概述和方法使用：

A:Arrays 类概述

* 针对数组进行操作的工具类。

* 提供了排序，查找等功能。

B:成员方法

* public static String toString(int[] a)

* public static void sort(int[] a)

* public static int binarySearch(int[] a,int key)

基本类型包装类的概述：

A:为什么会有基本类型包装类

* 将基本数据类型封装成对象的好处在于可以在对象中定义更多的功能方法操作该数据。

B:常用操作

* 常用的操作之一：用于基本数据类型与字符串之间的转换。

C:基本类型和包装类的对应

byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Integer 类的概述和构造方法：

A:Integer 类概述

* 通过 JDK 提供的 API，查看 Integer 类的说明

* Integer 类在对象中包装了一个基本类型 int 的值,

* 该类提供了多个方法，能在 int 类型和 String 类型之间互相转换，

* 还提供了处理 int 类型时非常有用的其他一些常量和方法

B:构造方法

* public Integer(int value)

* public Integer(String s)

C:案例演示

* 使用构造方法创建对象

String 和 int 类型的相互转换：

A:int -- String

* a:和""进行拼接

* b:public static String valueOf(int i)

* c:int -- Integer -- String(Integer 类的 toString 方法())

* d:public static String toString(int i)(Integer 类的静态方法)

B:String -- int

* a:String -- Integer -- int

* public static int parseInt(String s)

JDK5 的新特性自动装箱和拆箱：

A:JDK5 的新特性

* 自动装箱：把基本类型转换为包装类类型

* 自动拆箱：把包装类类型转换为基本类型

B:案例演示

* JDK5 的新特性自动装箱和拆箱

* Integer ii = 100;

* ii += 200;

C:注意事项

* 在使用时，Integer x = null;代码就会出现 NullPointerException。

* 建议先判断是否为 null，然后再使用。