●StringBuffer 的构造方法

StringBuffer 类的概述

通过 JDK 提供的 API

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

StringBuffer()

构造一个其中不带字符的字符串缓冲区，初始容量为 16 个字符。

```
package com.sxt.test;

public class Demo1 {
    public static void main(String[] args) {
        StringBuffer s = new StringBuffer();//从源代码可以看出,底层是 new byte[16]数组
       int len = s.capacity();//获取容量
       System.out.println(len);
        int a = s.length();//获取 StringBuffer 里实际面装了多少的元素
       System.out.println(a);
        StringBuffer s2 = s.append("你好");//append 方法往 StringBuffer 中添加数据,并且返回
StringBuffer 的对象本身
       System.out.println(s2);//说明底层重写了 Object 的 toString 方法
       System.out.println(s.hashCode());
        System.out.println(s2.hashCode());
        s.append("长沙 abcdef 你好长沙 ab");
        System.out.println(s);
        System.out.println(s.length());
        System.out.println(s.capacity());
        s.append("d");//当装入的数据超过初始容量,新的容量会翻倍,底层重新创建了一个 byte[34]数组
       System.out.println(s.length());
        System.out.println(s.capacity());

    }
}
```

StringBuffer(int capacity)

构造一个不带字符，但具有指定初始容量的字符串缓冲区。

```
package com.sxt.test;

public class Demo1 {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer(20);
        System.out.println("容量"+sb.capacity());
        System.out.println(sb.length());
        sb.append("abcdeabcdeabcdeabcdea");
        System.out.println("容量:"+sb.capacity());//超过指定的容量后,容量翻倍
        System.out.println("实际长度:"+sb.length());
```

```
    }
}
```

StringBuffer(String str)

构造一个字符串缓冲区，并将其内容初始化为指定的字符串内容。

```
package com.sxt.test;

public class Demo1 {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("abcd");//默认是16+装入数据的实际长度
       System.out.println(sb);
        System.out.println(sb.capacity());//20

    }
}
```

StringBuffer 的方法

Public int capacity():返回当前容量， 理论值

Public int length()：返回长度，(字符数),实际值

案列演示

构造方法和长度方法的使用

●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

```
package com.sxt.test;
public class Demo1 {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("abcd");//默认是16+装入数据的实际长度
       System.out.println(sb);
        System.out.println(sb.capacity());//20
        sb.insert(2,"z");//把z到索引为2的位置上
       System.out.println(sb);
    }
}
```

●StringBuffer 的删除功能

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到指定位置结束的内容，并返回本身

```
package com.sxt.test;

public class Demo1 {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("abcdefg");
        sb.deleteCharAt(3);
        System.out.println(sb);//abcefg
         sb.delete(0,3);//[ ) ,包含头,不包含尾
       System.out.println(sb);

    }
}
```

●StringBuffer 的替换和反转功能

public StringBuffer replace(int start,int end,String str);

从 start 开始到 end 用 str 替换

反转功能

public SringBuffer reverse();

字符串反转

```
package com.sxt.test;

public class Demo1 {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("长沙是个宜居的城市,幸福指数高");
        sb.replace(0,2,"广州");// [ ) ,包头不包尾
       System.out.println(sb);
        sb.reverse();//把 sb 中所有的数据反转,并返回 sb 对象本身
       System.out.println(sb);

    }
}
```

●StringBuffer 的截取功能

String Substring(int start);

从指定位置截取到末尾

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

注意：返回不再是 StringBuffer 本身了

```
package com.sxt.test;

public class Demo1 {
```

```
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("长沙是个宜居的城市,幸福指数高");
        String s = sb.substring(2);//从索引 2 开始截取到末尾
       System.out.println(s);
        String s2 = sb.substring(2,6);//从这里面截取:长沙是个宜居的城市,幸福指数高
       System.out.println(s2);
    }
}
```

●StringBuffer 与 String 转换

String 对象 -->StringBuffer 对象

通过构造方法 StringBuffer(String s);

通过 append（）方法

StringBuffer --> String

通过构造方法

通过 toString()方法

通过 subString(0,length);

```
package com.sxt.test;

public class Demo1 {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("长沙是个宜居的城市,幸福指数高");
       //StringBuffer 转 String 类型
      String s1 = sb.toString();
        System.out.println(s1);
        String s2 = sb.substring(0);
        System.out.println(s2);

        //String 转 StringBuffer
        String ss = "我爱长沙星城";
        StringBuffer sb2 = new StringBuffer(ss);
        StringBuffer sb3 = new StringBuffer();
        StringBuffer ss2 = sb3.append(ss);

    }
}
```

案列：需求

把数组中的数据按照指定的格式拼接成一个字符串

举例：

Int[] arr ={1,2,3};

输出结果：”[1,2,3]”

用 StringBuffer 的功能实现

```
package com.sxt.test;

public class Demo1 {
    public static void main(String[] args) {

        /*
             int[] arr ={1,2,3};
          输出结果：”[1,2,3]”
         */
        int[] arr ={1,2,3};
        StringBuffer sb = new StringBuffer();
        sb.append("[");
        for (int i = 0; i <arr.length ; i++) {
            int n = arr[i];
            sb.append(n).append(",");
        }
        sb.replace(sb.length()-1,sb.length(),"");
        sb.append("]");

        System.out.println(sb);

        System.out.println("---------------");

        //StringBuffer 可以添加任何数据类型,包括对象
       sb.append(new Person("小明",19));
        System.out.println(sb);
    }
}
class Person{
    String name;
    int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

StringBuffer 与 StringBuilder 是同门兄弟类,功能完全一致

区别:前者线程安全,效率略低. 后者线程不安全,效率高

StringBuffer， StringBuilder 速度测试

```
package com.sxt.test;

public class Demo1 {
    public static void main(String[] args) {
        long time3 = System.currentTimeMillis();
        StringBuilder sb2 = new StringBuilder();
        for (int i = 0; i <100000000 ; i++) {
            sb2.append(i);
        }
        long time4 = System.currentTimeMillis();
        System.out.println("StringBuilder 用时:"+(time4 - time3));

        long time1 = System.currentTimeMillis();
        StringBuffer sb1 = new StringBuffer();
        for (int i = 0; i <100000000 ; i++) {
            sb1.append(i);
        }
        long time2 = System.currentTimeMillis();
        System.out.println("StringBuffer 用时:"+(time2 - time1));



    }
}
```