**public** **class** StringDemo {

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

/\*String类特点：

\* 字符串对象 一旦被初始化就不会被改变

*stringDemo2*();

}

**private** **static** **void** stringDemo2() {

String s="abc";/

s="nba";/

System.*out*.println(“s:”+s);//这是两个字符串对象

一开始指向abc,现在指向nba,但是abc没有改变，我们说的字符串对象一旦初始化就不能改变，s就是一个引用型变量，它可以指向这个 也可以指向那个，但是abc并没有改变字符串一旦被初始化，就不可以被改变应该是 "abc"这个对象一直在内存中，而s是可以改变的，s是可以指向另一个对象的

}

**private** **static** **void** stringDemo2() {

String s="abc";//创建一个字符串对象在常量池中

String s1=**new** String("abc");//创建两个对象,一个new 一个字符串对象在堆内存中

System.*out*.println(s==s1);//比较的是是否是同一个地址？？false

System.*out*.println(s.equals(s1));//true 其实就是比较字符串内容

}

/\*演示字符串定义的第一种方式：并明确字符串常量池的特点

\* 池中没有就建立，有就直接用

\* \*/

**public** **static** **void** stringDemo1() {

String s="abc";//"abc"存储在字符串常量池中，如果没有就创建

String s1="abc";

System.*out*.println(s==s1);//true

}

**String类**

**public** **class** StringConstructorDemo {

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

*StringConstructorDemo2*();

}

**private** **static** **void** StringConstructorDemo2() {

**char**[] arr={'w','a','p','q','x'};

String s=**new** String(arr);//将字符数组变化成字符串

System.*out*.println("s:"+s);//s:wapqx

String s1=**new** String(arr,1,3);//获取数组的第1个到第3个角标

System.*out*.println("s1:"+s1);//s1:apq

}

**public** **static** **void** StringConstructorDemo() {

String s=**new** String();//等效String s="" 不等效 String s=null;

**byte**[] arr={65,66,67,68};

String s1=**new** String(arr);//将字节数组变成字符串

System.*out*.println("s1:"+s1);

}

}

d

**public** **class** StringMethodDemo {

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

/\*

\* 按照面向对象的思想对字符串进行功能分类。

\* "abcd"

\*

\* 1,获取：

\* 1.1 获取字符串中字符的个数(长度).

\* int length();

\* 1.2 根据位置获取字符。

\* char charAt(int index);

\* 1.3 根据字符获取在字符串中的第一次出现的位置.

\* int indexOf(int ch)

\* int indexOf(int ch,int fromIndex):从指定位置进行ch的查找第一次出现位置

\* int indexOf(String str);

\* int indexOf(String str,int fromIndex);

\* 根据字符串获取在字符串中的第一次出现的位置.

\* int lastIndexOf(int ch)

\* int lastIndexOf(int ch,int fromIndex):从指定位置进行ch的查找第一次出现位置

\* int lastIndexOf(String str);

\* int lastIndexOf(String str,int fromIndex);

\* 1.4 获取字符串中一部分字符串。也叫子串.

\* String substring(int beginIndex, int endIndex)//包含begin 不包含end 。

\* String substring(int beginIndex);

\* 2，转换。

\* 2.1 将字符串变成字符串数组(字符串的切割)

\* String[] split(String regex):涉及到正则表达式.

\* 2.2 将字符串变成字符数组。

\* char[] toCharArray();

\* 2.3 将字符串变成字节数组。

\* byte[] getBytes();

\* 2.4 将字符串中的字母转成大小写。

\* String toUpperCase():大写

\* String toLowerCase():小写

\* 2.5 将字符串中的内容进行替换

\* String replace(char oldch,char newch);

\* String replace(String s1,String s2);

\* 2.6 将字符串两端的空格去除。

\* String trim();

\* 2.7 将字符串进行连接 。

\* String concat(string);

\* 3，判断

\* 3.1 两个字符串内容是否相同啊？

\* boolean equals(Object obj);

\* boolean equalsIgnoreCase(string str);忽略大写比较字符串内容。

\* 3.2 字符串中是否包含指定字符串？

\* boolean contains(string str);

\* 3.3 字符串是否以指定字符串开头。是否以指定字符串结尾。

\* boolean startsWith(string);

\* boolean endsWith(string);

}

**public** **static** **void** StringMethodDemo\_1() {

String s="abcdae";

System.*out*.println("字符串的长度:"+s.length());//5

System.*out*.println("字符串的位置："+s.charAt(2));//c

System.*out*.println("字母在字符串的小标:"+s.indexOf('b'));//1，如果没有找到该字符，就用-1表示

System.*out*.println("字母最后一次出现的位置："+s.lastIndexOf("a"));//4

System.*out*.println("substring："+s.substring(2));//cdae

System.*out*.println("substring："+s.substring(2,4));//cd 包含begin 不包含end

System.*out*.println("substring："+s.substring(2,s.length()));//cdae

}

**public** **static** **void** StringMethodDemo\_2() {

//2.1 将字符串变成字符串数组(字符串的切割)

String s="张三,李四,王五";

String[] arr=s.split(",");

**for**(**int** i=0;i<arr.length;i++){

System.*out*.println(arr[i]);

}

//注意:String[] arr=s.split(",");如果是.的话 需要转义

//String[] arr=s.split("\\.");点是正则里面的 需要进行转义成普通的点

//2.2 将字符串变成字符数组。char[] toCharArray();

String s1="张三,李四,王五";

**char**[] chs = s1.toCharArray();

**for**(**int** j=0;j<chs.length;j++){

System.*out*.println(chs[j]);

}一个一个打印

//2.3 将字符串变成字节数组。byte[] getBytes();

String s2="ab你";//4个字节

**byte**[] bytes = s2.getBytes();

System.*out*.println(bytes.length);// 4 gbk中 一个汉字两个字节

**for**(**int** i=0;i<bytes.length;i++){

System.*out*.println(bytes[i]);

}

String s3="你你你";

**byte**[] b=s3.getBytes();

System.*out*.println(b.length);//6

//2.4 将字符串中的字母转成大小写。

String s4="abc";

System.*out*.println(s4.toUpperCase());//ABC 把abc转化成大写的

String s5="WUST";

System.*out*.println(s5.toLowerCase());//wust 把WUST转化成小写

//字符替换

String s6="javaEE";

System.*out*.println(s6.replace('a', 'o'));//jovoEE

//字符串替换

System.*out*.println(s6.replace(s6, "wust"));//wust

//去除字符串两端的空格字符

String s7="-"+" ab c ".trim()+"-";

System.*out*.println(s7);

//2.7 将字符串进行连接 。String concat(string);

System.*out*.println("Wust".concat("zdy"));//Wustzdy

}

**public** **static** **void** StringMethodDemo\_3() {

String s="abc";

System.*out*.println(s.equals("ABC".toLowerCase()));//true

System.*out*.println(s.equalsIgnoreCase("ABC"));//true 忽略大小写

System.*out*.println(s.contains("bc"));//true 是否包含bc

String str="ArrayDemo.java";

System.*out*.println(str.startsWith("Array"));//以Array开头的

System.*out*.println(str.endsWith(".java"));//以。Java为结尾的

System.*out*.println(str.concat("Demo"));//包含Demo的文件

}

**public** **static** **void** StringMethodDemo\_4() {

}

}

**public** **class** StringBufferDemo {

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

/\*StringBuffer:就是字符串缓冲区

\* 用于存储数据的容器

\*特点：

\*1，长度是可变的

\*2，可以存储不同类型的数据

\*3,最终要转化成字符串进行使用

\*

\*既然是一个容器对象。应该具备什么功能？？

\*1,添加

\* StringBuffer append(data)

\*2,删除

\* StringBuffer delete(start,end):包含头，不包含尾

\* StringBuffer deleteCharAt(int index):删除指定位置的元素

\*

\*/

//bufferMethodDemo1();

*bufferMethodDemo2*();

}

**public** **static** **void** bufferMethodDemo1(){

//创建缓冲区对象

StringBuffer sb=**new** StringBuffer();

StringBuffer s1 = sb.append(4);

sb.append(4).append(**false**).append("hahh");

sb.insert(1, "hahh");

System.*out*.println(sb);// 4hahh4falsehahh

System.*out*.println(s1);// 4hahh4falsehahh

}

**public** **static** **void** bufferMethodDemo2(){

StringBuffer sb=**new** StringBuffer("abce");

//sb.delete(1, 3);//ae

//清空缓冲区

sb.delete(0, sb.length());//

System.*out*.println(sb);

}

**public** **class** StringBuilderDemo {

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

/\*

\* jdk1.5以后出现了功能和StringBuffer一模一样的对象。就是StringBuilder

\*

\* 不同的是：

\* StringBuffer是线程同步的。通常用于多线程。

\* StringBuilder是线程不同步的。通常用于单线程。 它的出现提高效率。

\*

\* jdk升级：

\* 1，简化书写。

\* 2，提高效率。

\* 3，增加安全性。

\*/

//把一个int数组变成字符串

**int**[] arr={3,1,5,3,8};

// /String s=arrayToString(arr);

String s=*arrayToString\_1*(arr);

System.*out*.println(s);

}

**public** **static** String arrayToString(**int**[] arr){

String str="[";

**for**(**int** i=0;i<arr.length;i++){

**if**(i!=arr.length-1)

str+=arr[i]+",";

**else** {

str+=arr[i]+"]";

}

}

**return** str;

}

//效率高,一次性读入缓冲区，一次性变成字符串

**public** **static** String arrayToString\_1(**int**[] arr){

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

sb.append("[");

**for**(**int** i=0;i<arr.length;i++){

**if**(i!=arr.length-1)

sb.append(arr[i]+",");

**else** {

sb.append(arr[i]+"]");

}

}

**return** sb.toString();

}

}

/\*

\* 1，给定一个字符串数组。按照字典顺序进行从小到大的排序。

\* {"nba","abc","cba","zz","qq","haha"}

\*

\* 思路：

\* 1,对数组排序。可以用选择，冒泡都行。

\* 2,for嵌套和比较以及换位。

\* 3,问题：以前排的是整数，比较用的比较运算符，可是现在是字符串对象。

\* 字符串对象怎么比较呢？爽了，对象中提供了用于字符串对象比较的功能。

\*/

**public** **class** StringTest\_1 {

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

String[] arr={"nba","abc","cba","zz","qq","haha"};

*printArray*(arr);

*sortString*(arr);

*printArray*(arr);

}

**private** **static** **void** sortString(String[] arr) {

**for**(**int** i=0;i<arr.length-1;i++){

**for**(**int** j=i+1;j<arr.length;j++){

**if**(arr[i].compareTo(arr[j])>0){

*swap*(arr,i,j);

}

}

}

}

**private** **static** **void** swap(String[] arr, **int** i, **int** j) {

String temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

**private** **static** **void** printArray(String[] arr) {

System.*out*.print("[");

**for**(**int** i=0;i<arr.length;i++){

**if**(i!=arr.length-1)

System.*out*.print(arr[i]+",");

**else** {

System.*out*.println(arr[i]+"]");

}

}

}

}

/\*

\* 2，一个子串在整串中出现的次数。

\* "nbaernbatynbauinbaopnba"

\* 思路：

\* 1，要找的子串是否存在，如果存在获取其出现的位置。这个可以使用indexOf完成。

\* 2，如果找到了，那么就记录出现的位置并在剩余的字符串中继续查找该子串，

\* 而剩余字符串的起始位是出现位置+子串的长度.

\* 3,以此类推，通过循环完成查找，如果找不到就是-1，并对 每次找到用计数器记录。

\*

\*/

**public** **class** StringTest\_2 {

/\*\*

\* **@param** args

\*/

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

String str = "nbaernbatnbaynbauinbaopnba";

String key = "nba";

**int** count = *getKeyStringCount\_2*(str,key);

System.*out*.println("count="+count);

}

**public** **static** **int** getKeyStringCount\_2(String str, String key) {

**int** count = 0;

**int** index = 0;

**while**((index = str.indexOf(key,index))!=-1){

index = index + key.length();

count++;

}

**return** count;

}

/\*\*

\* 获取子串在整串中出现的次数。

\* **@param** str

\* **@param** key

\* **@return**

\*/

**public** **static** **int** getKeyStringCount(String str, String key) {

//1,定义计数器。

**int** count = 0;

//2，定义变量记录key出现的位置。

**int** index = 0;

**while**((index = str.indexOf(key))!=-1){

str = str.substring(index+key.length());

count++;

}

**return** count;

}

}

/\*

\* 3，两个字符串中最大相同的子串。

\* "qwerabcdtyuiop"

\* "xcabcdvbn"

\*

\* 思路：

\* 1，既然取得是最大子串，先看短的那个字符串是否在长的那个字符串中。

\* 如果存在，短的那个字符串就是最大子串。

\* 2，如果不是呢，那么就将短的那个子串进行长度递减的方式去子串，去长串中判断是否存在。

\* 如果存在就已找到，就不用在找了。

\*/

**public** **class** StringTest\_3 {

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

String s1 = "qwerabcdtyuiabcdeop";

String s2 = "xcabcdvabcdebn";

String s = *getMaxSubstring*(s2, s1);

System.*out*.println("s=" + s);

}

**public** **static** String getMaxSubstring(String s1, String s2) {

String max = **null**,min = **null**;

max = (s1.length()>s2.length())?s1:s2;

min = max.equals(s1)?s2:s1;

System.*out*.println("max="+max);

System.*out*.println("min="+min);

**for** (**int** i = 0; i < min.length(); i++) {

**for**(**int** a = 0,b = min.length()-i; b != min.length()+1; a++,b++){

String sub = min.substring(a, b);

**if**(max.contains(sub))

**return** sub;

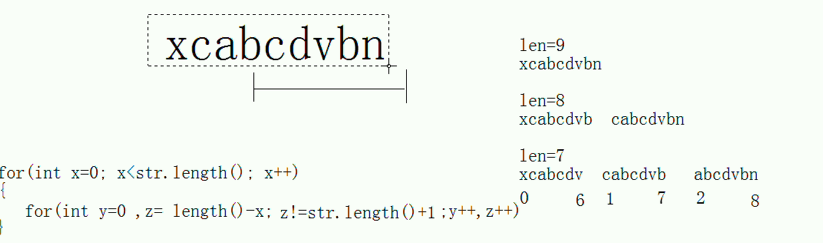
}

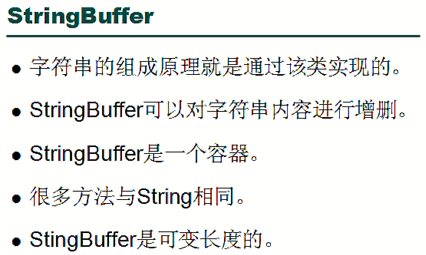
}

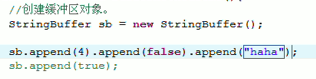
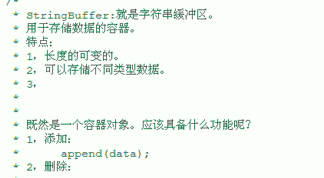
**return** **null**;

}

}







**sb.append(4).append(false).append("haha");//4falsehaha**

**sb.insert(1, "haha");//4hahafalsehaha在指定位置上插入haha**

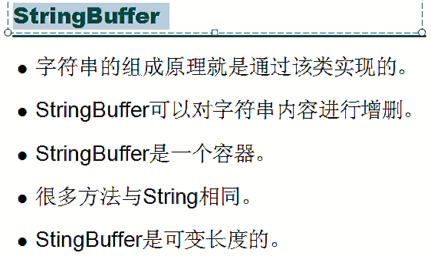
**StringBuffer sb = new StringBuffer("abce");**

**sb.delete(1, 3);//ae 包含头不包含尾**

**//清空缓冲区。**

**sb.delete(0,sb.length());**

**sb = new StringBuffer();**





\* jdk1.5以后出现了功能和StringBuffer一模一样的对象。就是StringBuilder

\* 不同的是：

\* StringBuffer是线程同步的。通常用于多线程。

\* StringBuilder是线程不同步的。通常用于单线程。 它的出现提高效率。不需要锁

\* jdk升级：

\* 1，简化书写。

\* 2，提高效率。

\* 3，增加安全性。StringBuffer 同步安全 StringBuilder不同步不安全

public static void main(String[] args) { \* 将一个int数组变成字符串。

int[] arr = {3,1,5,3,8};String s = arrayToString\_2(arr);System.out.println(s);

}

public static String arrayToString\_2(int[] arr){

StringBuilder sb = new StringBuilder();sb.append("[");

for (int i = 0; i < arr.length; i++) {

if(i!=arr.length-1)

sb.append(arr[i]+", ");

else

sb.append(arr[i]+"]");

}

return sb.toString();

}

public static String arrayToString(int[] arr){

String str = "[";

for (int i = 0; i < arr.length; i++) {

if(i!=arr.length-1)

str+=arr[i]+", ";

else

str+=arr[i]+"]";

}

return str;

}

但是这时候效率比较慢，我添加对象，需要判断锁，删除的时候又需要判断一次锁，如果是单线程的话，就拿删除对象来说，删一次，判断一次锁，删一次，又需要判断一次锁，这就导致效率比较慢

**所以单线程用StringBuilder,**线程不同步的,出现提高效率。不需要锁

**多线程用StringBuffer,**线程同步

class StringBuffer jdk1.0{

object lock;

public StirngBuffer append(int x){

}

public StringBuffer delete(int start,int end){

}

}

创建一个StringBuffer对象，一个线程往里面添加对象，一个线程往里面删除对象，这个时候就会出现线程安全隐患，所以我们要在添加的时候不能删除，删除的时候不能添加，所以这时候不安全 ，要枷锁，添加和删除要进行同步

class StringBuffer jdk1.0{

object lock;

public StirngBuffer append(int x){

synchronized(lock)

{

}

}

public synchronized StringBuffer delete(int start,int end){

synchronized(lock)

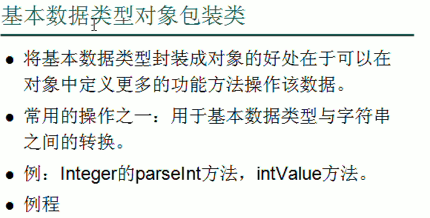
{

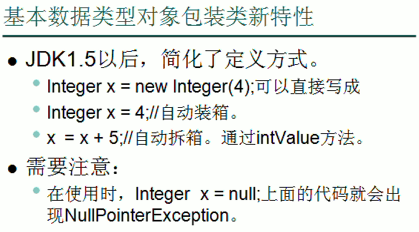
}

}

}

**为什么StringBuffer是同步，线程安全的？**





/\*

\* 基本数据类型对象包装类。

\* 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行为丰富了该数据的操作。

\* 用于描述该对象的类就称为基本数据类型对象包装类。

\* byte Byte

\* short Short

\* int Integer

\* long Long

\* float Float

\* double Double

\* char Character

\* boolean Boolean

\*

\* 该包装对象主要用基本类型和字符串之间的转换。

\*

**\* 基本类型--->字符串**

\* 1,基本类型数值+"" //1+"22"=122

\* 2,用String类中的静态方法valueOf(基本类型数值);//String.valueOf(123)="123"

\* 3,用Integer的静态方法valueOf(基本类型数值);

\*

**\* 字符串--->基本类型**

\* 1,使用包装类中的静态方法 xxx parseXxx("xxx类型的字符串");\*\*\*\*\*

\* int parseInt("intstring");// Int a=Integer.parseInt("123");//123

\* long parseLong("longstring");//Long b=Long.parseLong("")

\* boolean parseBoolean("booleanstring");//Boolean c=Boolean.parseBoolean("true")

\* 只有Character没有parse方法

\* 2,如果字符串被Integer进行对象的封装。

\* 可使用另一个非静态的方法，intValue();

\* 将一个Integer对象转成基本数据类型值。

\* Integer i = new Integer("123");

System.out.println(i.intValue());//123 将一个Integer对象转成基本数据类型值。

2,如果字符串被Integer进行对象的封装。

\* 可使用另一个非静态的方法，intValue();

\* 将一个Integer对象转成基本数据类型值。

\* Integer i = new Integer("123");

System.out.println(i.intValue());//123 将一个Integer对象转成基本数据类型值。

\* /\*

\* 整数具备不同的进制体现。

\*

\* 十进制-->其他进制。

\* toBinaryString

\* toOctalString

\* toHexString

\*

\* 其他进制-->十进制。

\* parseInt("string",radix)

\*

\*/

System.*out*.println(Integer.*MAX\_VALUE*);//(2^31)-1

System.*out*.println(Integer.*MIN\_VALUE*);//-(2^31)

System.*out*.println(Integer.*toBinaryString*(6));//6对应的二进制 110

**int** num=4;

Integer integer = **new** Integer(num);//基本整数类型变成对象

//如何把String类型变成基本数据类型

**int** x = Integer.*parseInt*("123");//123

System.*out*.println(Integer.*parseInt*("123")+1);//124 //改方法必须要传整数类型的字符串

Integer i = **new** Integer("123");

System.*out*.println(i.intValue());//123 将一个Integer对象转成基本数据类型值。

//十进制-->其他进制。

System.*out*.println(Integer.*toBinaryString*(60));// 十进制-->二进制111100

System.*out*.println(Integer.*toOctalString*(60));//十进制-->八进制 74

System.*out*.println(Integer.*toHexString*(60));//二进制-->十六进制 3c

System.*out*.println(Integer.*toString*(60,16));//把十进制的数60 转化后成16进制的数

//其他进制-->十进制。

System.*out*.println(Integer.*parseInt*("3c",16));//60

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(a.compareTo(b));//0

//a.compareTo(b)=0,a==b

//a.compareTo(b)=-1,a<b

//a.compareTo(b)=1,a>b

}

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

**int** num = 4;

Integer in=**new** Integer(4);//1.4之前

Integer i=4;//1.5之后，自动将整数转化成对象

i=i+6;//i=new Integer(i.intValue()+6),//i.intValue()自动拆箱

Integer a = **new** Integer(128);

Integer b = **new** Integer(128);

System.*out*.println(a==b);//false

System.*out*.println(a.equals(b));//true

Integer x = 127;//jdk1.5以后，自动装箱，如果装箱的是一个字节，那么该数据会被共享不会重新开辟空间。

Integer y = 127;

System.*out*.println(x==y);//true

System.*out*.println(x.equals(y));//true

Integer x1 = 128;//jdk1.5以后，自动装箱，不是一个字节，那么该数据需要重新开辟空间。

Integer y1 = 128;

System.*out*.println(x==y);//false

System.*out*.println(x.equals(y));//true

}