常用类

面向对象程序设计

第 11 讲 OOP 编程 - 常用类

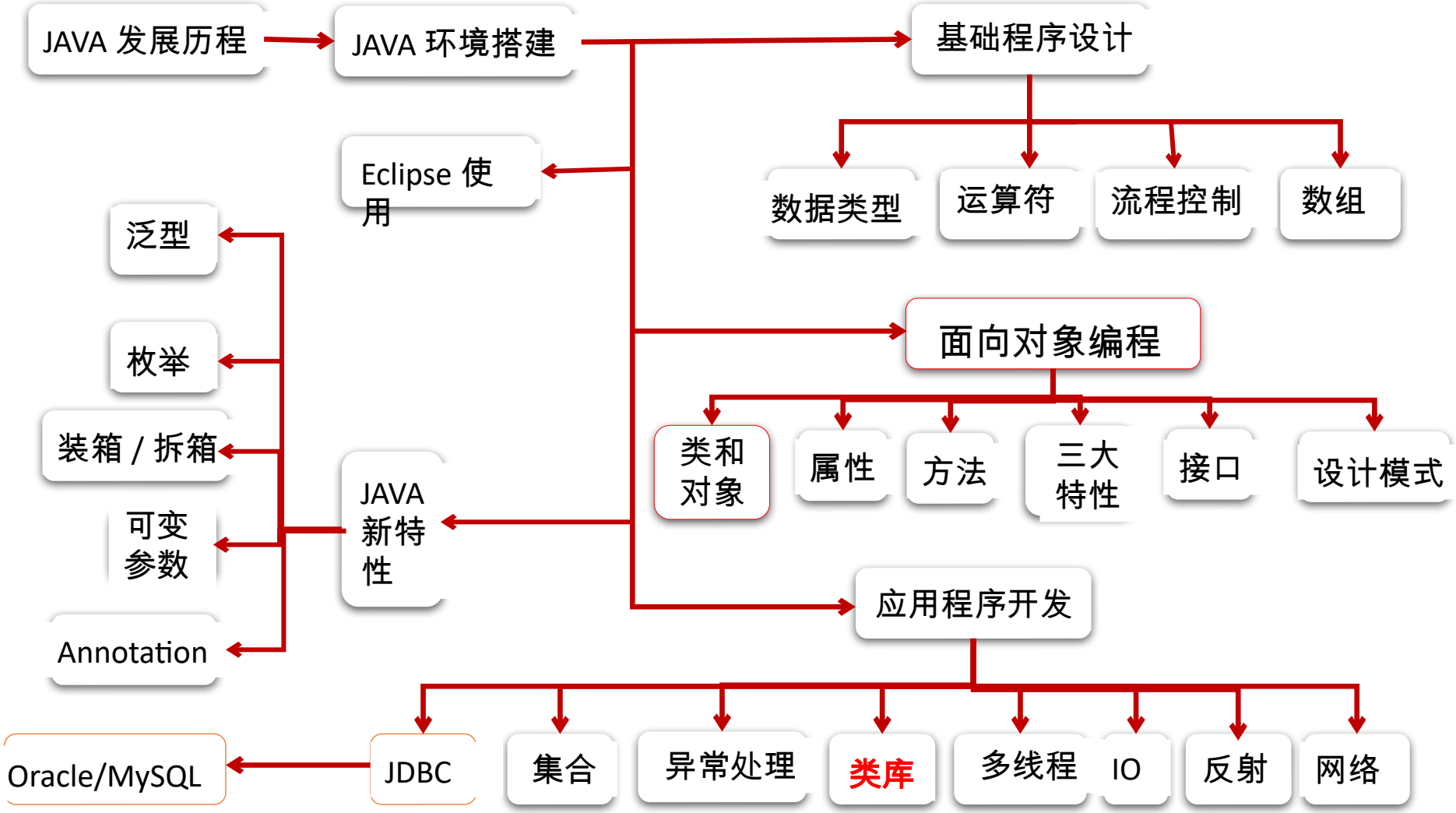
刘进

2230652597@qq.com

OOP 教辅 2025 秋季 QQ 群：305915615

群名称：OOP教辅2022秋季
群　号：305915615

Java 基础知识图解


JAVA 发展历程
JAVA 环境搭建
基础程序设计
Eclipse 使用
数据类型
运算符
流程控制
数组
泛型
枚举
装箱 / 拆箱
可变参数
Annotation
JAVA 新特性
面向对象编程
类和对象
属性
方法
三大特性
接口
设计模式
应用程序开发
Oracle/MySQL
JDBC
集合
异常处理
类库
多线程
IO
反射
网络

主要内容

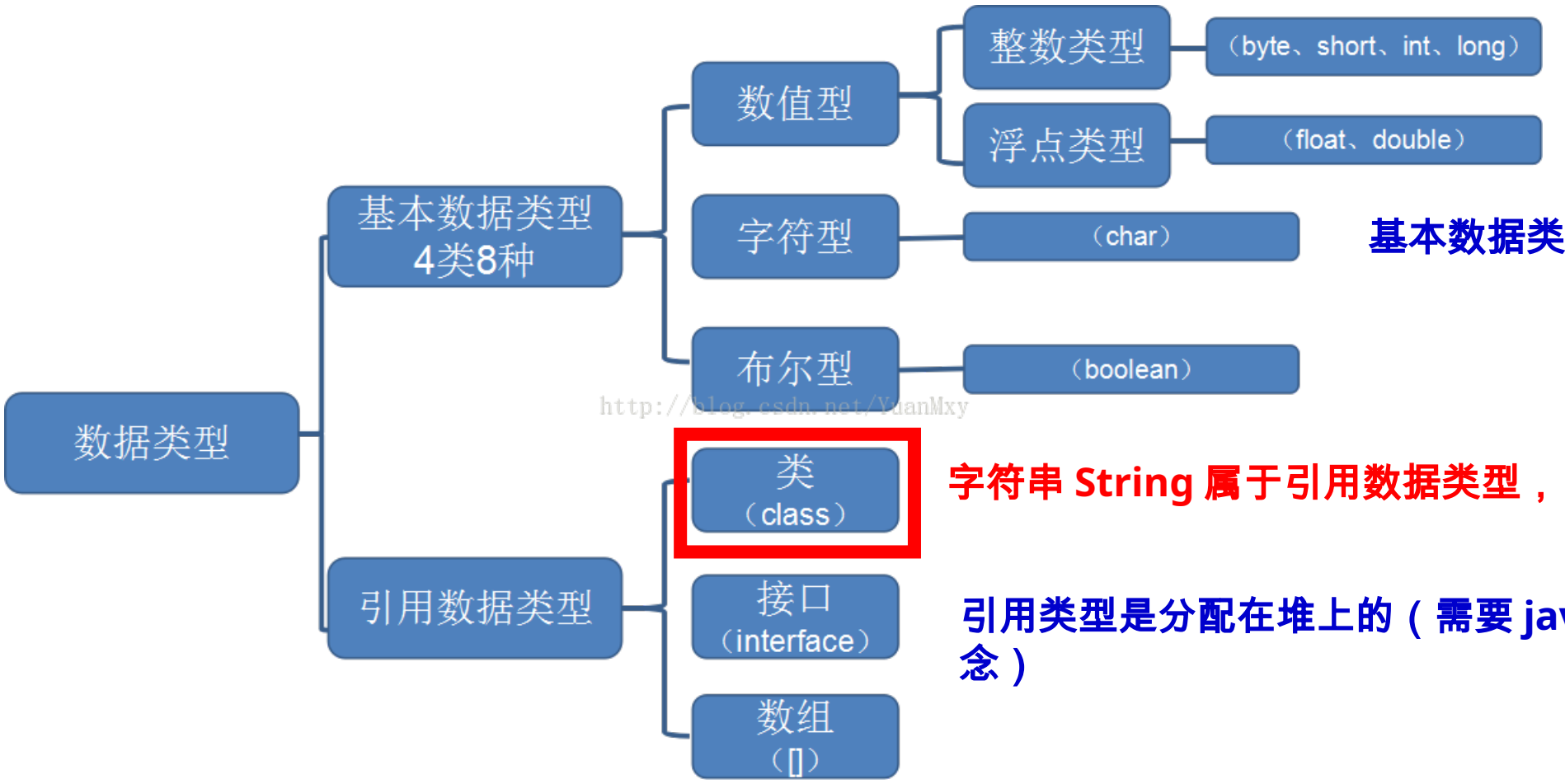
内容

- String 类
- StringBuffer 类
- StringBuilder 类
- System 类
- Date 类
- SimpleDateFormat 类
- Calendar 类
- Math 类
- BigInteger 类与 BigDecimal 类

一、String 类

String 类型和基本的数据类型有什么区别：

区别
注意 S 是大写的



基本数据类型是分配在栈上的

字符串 String 属于引用数据类型，String 是一个类

引用类型是分配在堆上的（需要 java 中的栈、堆概念）

基本数据类型和引用类型的区别

基本数据类型的存储原理

所有的简单数据类型不存在“引用”的概念，基本数据类型都是直接存储在内存中的内存栈上的，数据本身的值就是存储在栈空间里面，而 Java 语言里面八种数据类型是这种存储模型；

String类是final类，可以从String类的定义来看出。String类的开头定义如下：

String 的原始定义

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence
    /** The value is used for character storage. */
    private final char value[];

    /** Cache the hash code for the string */
    private int hash; // Default to 0
```

1 ）String 类是 final 类，也即意味着 String 类不能被继承，并且它的成员方法都默认为 final 方法。在 Java 中，**被 final 修饰的类是不允许被继承的，并且该类中的成员方法都默认为 final 方法**。

2 ）上面列举出了 String 类中所有的成员属性，从上面可以看出 String 类其实是通过 char 数组来保存字符串的。

引用数据类型的存储原理

引用类型继承于 Object 类（也是引用类型）都是按照 Java 里面存储对象的内存模型来进行数据存储的，使用 Java 内存堆和内存栈来进行这种类型的数据存储，简单地讲，**“引用”是存储在有序的内存栈上的，而对象本身的值存储在内存堆上的**

基本数据类型和引用类型的区别

一、字符串相关类

区别

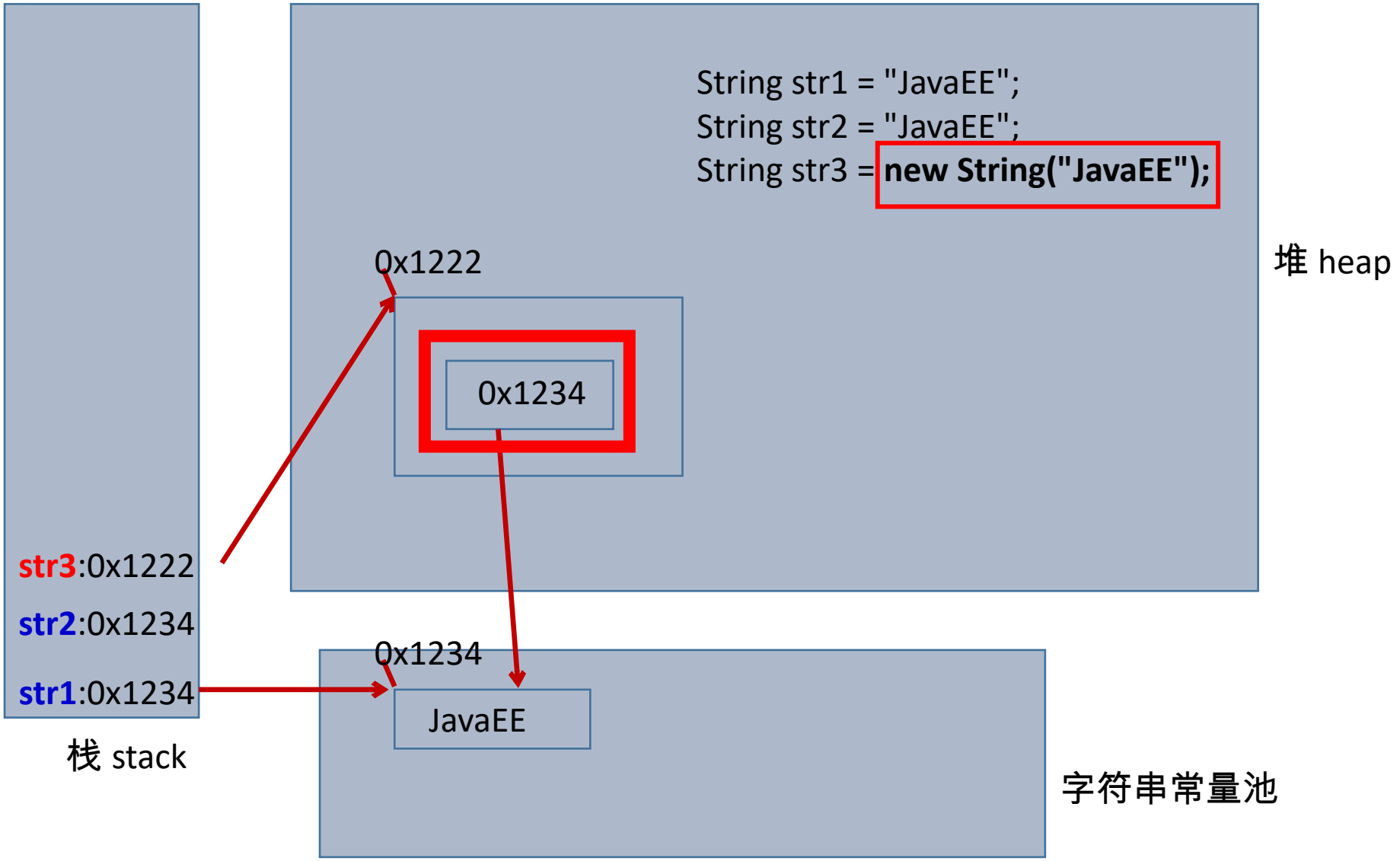
String 类：构造字符串对象

- 常量对象：字符串常量对象是用双引号括起的字符序列。

  例如：" 你好 " 、 "12.97" 、 "boy" 等。

- 字符串的字符使用 Unicode 字符编码，一个字符占两个字节

- String 类较常用构造方法：
  - String s1 = new String();
  - String s2 = new String(String original);
  - String s3 = new String(char[] a);
  - String s4 = new String(char[] a,int startIndex,int count)
  - ◆ String str = “abc”; 与 String str1 = new String(“abc”); 的区别？


String str1 = "JavaEE";
String str2 = "JavaEE";
String str3 = new String("JavaEE");
堆 heap
0x1222
0x1234
str3:0x1222
str2:0x1234
str1:0x1234
栈 stack
0x1234
JavaEE
字符串常量池

字符串的特性

区别

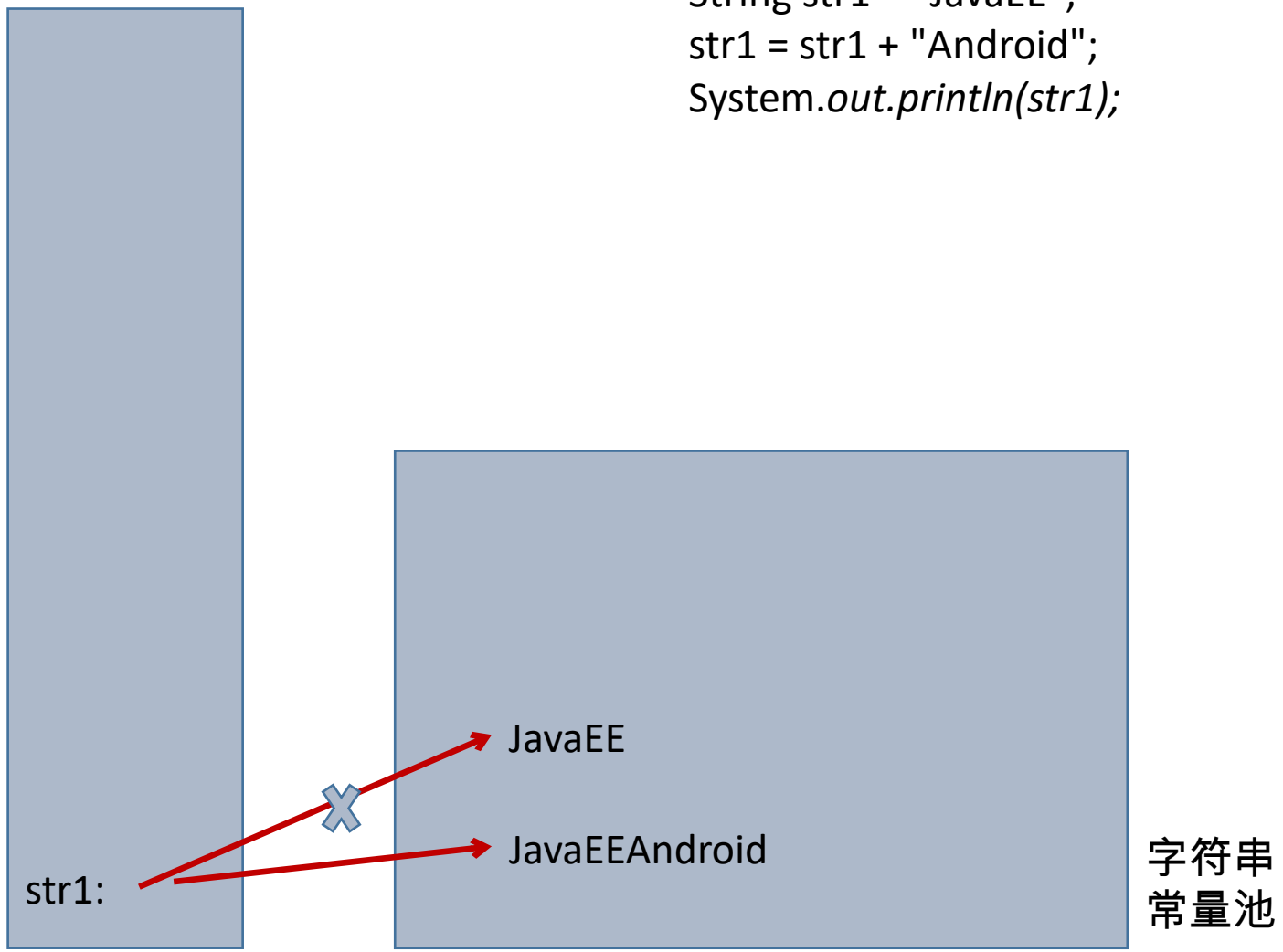
- String 是一个 final 类，代表不可变的字符序列
- 字符串是不可变的。一个字符串对象一旦被配置，其内容是不可变的。

判断：

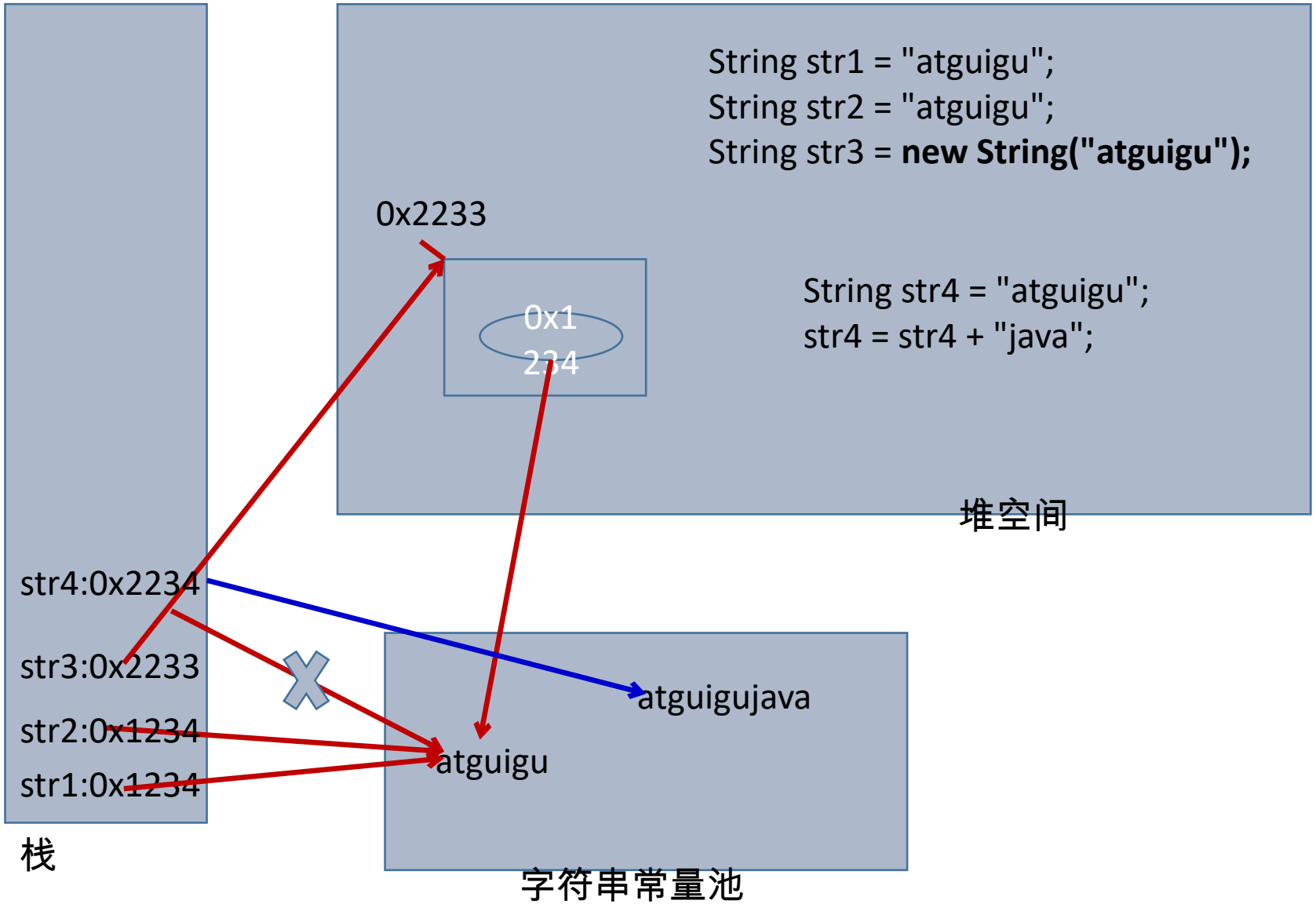
```
String s1 = "atguigu";
String s2 = "java";
String s4 = "java";
String s3 = new String("java");
System.out.println(s2 == s3);
System.out.println(s2 == s4);
System.out.println(s2.equals(s3));
```

```
String s5 = "atguigujava";
String s6 = (s1 + s2).intern();
System.out.println(s5 == s6);
System.out.println(s5.equals(s6));
```

```
String str1 = "JavaEE";
str1 = str1 + "Android";
System.out.println(str1);
```



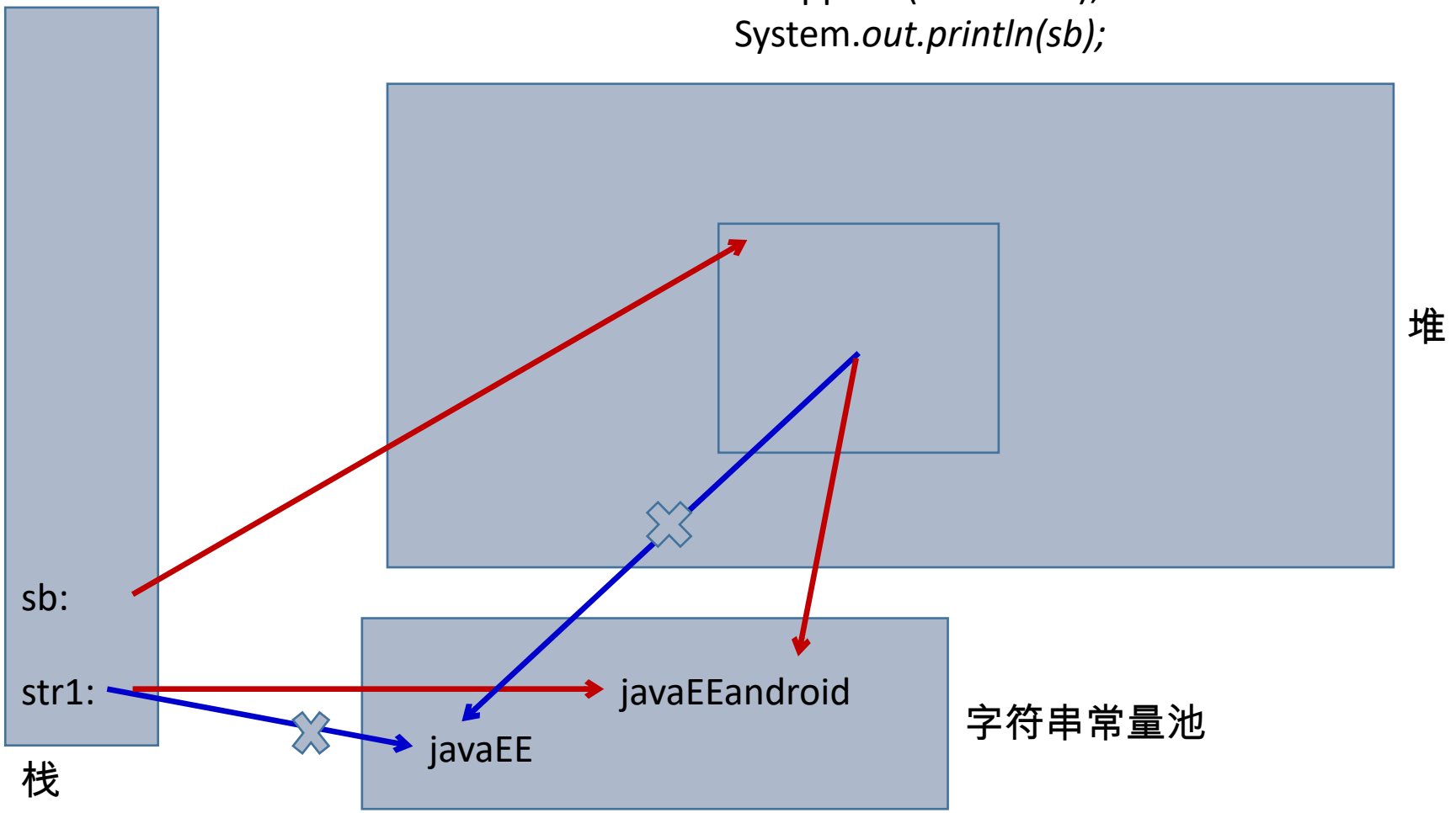
区别

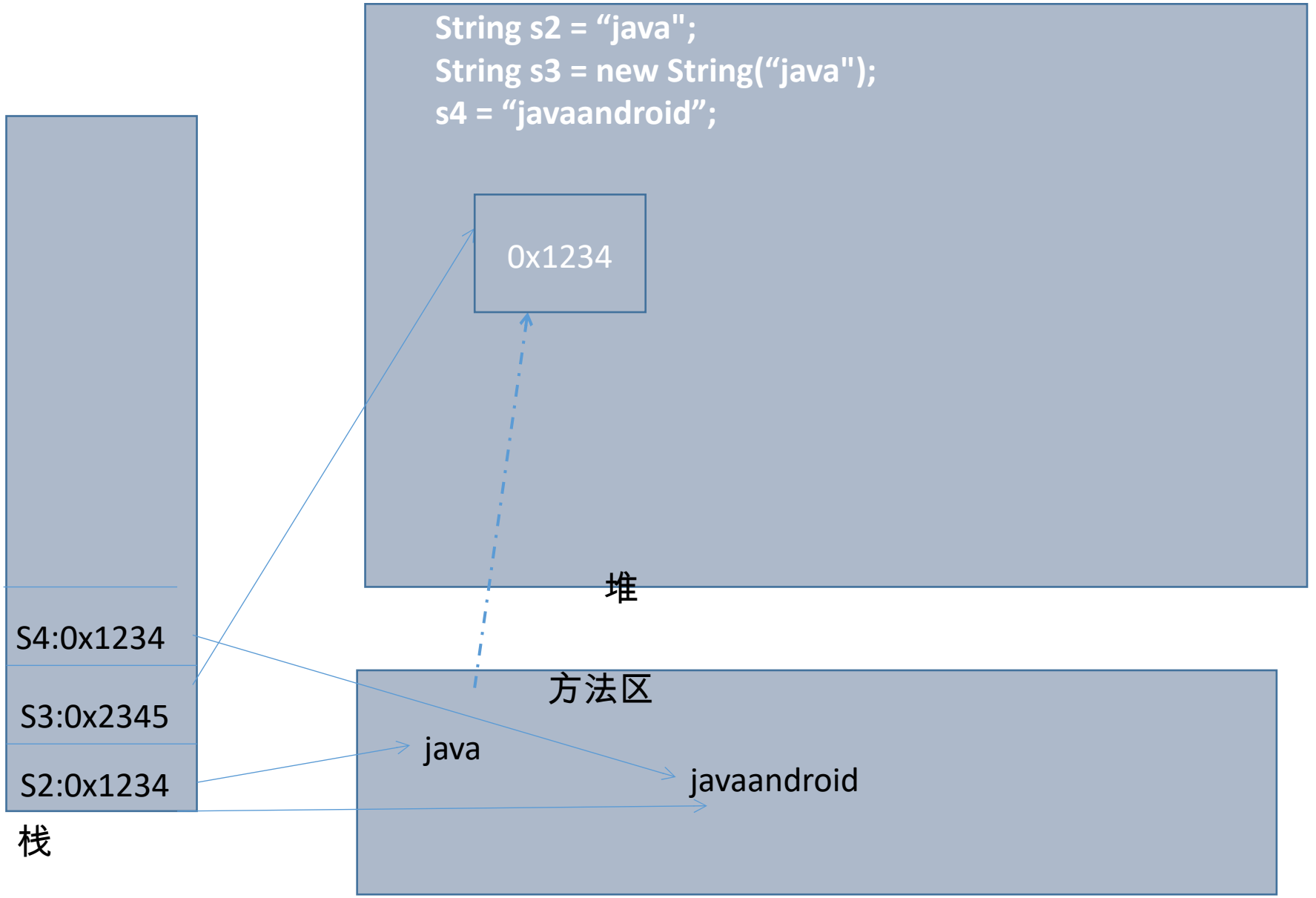

String str1 = "atguigu";
String str2 = "atguigu";
String str3 = new String("atguigu");
0x2233
0x1
234
String str4 = "atguigu";
str4 = str4 + "java";
堆空间
str4:0x2234
str3:0x2233
str2:0x1234
str1:0x1234
atguigujava
atguigu
栈
字符串常量池

区别

String str1 = "javaEE";
str1 = "javaEEAndroid";

StringBuffer sb = **new StringBuffer("javaEE");**
sb.append("android");
System.*out.println(sb);*




String s2 = “java";
String s3 = new String(“java");
s4 = “javaandroid”;
0x1234
堆
S4:0x1234
S3:0x2345
S2:0x1234
栈
方法区
java
javaandroid

字符串对象操作

检索操作

- public int length()
- public char charAt(int index)
- public boolean equals(Object anObject)
- public int compareTo(String anotherString)
- public int indexOf(String s)
- public int indexOf(String s ,int startpoint)
- public int lastIndexOf(String s)
- public int lastIndexOf(String s ,int startpoint)
- public boolean startsWith(String prefix)
- public boolean endsWith(String suffix)
- public boolean regionMatches(int firstStart,String other,int otherStart ,int length)

字符串对象修改

修改区别

- public String substring(int startpoint)
- public String substring(int start,int end)
- pubic String replace(char oldChar,char newChar)
- public String replaceAll(String old,String new)
- public String trim()
- public String concat(String str)
- public String[] split(String regex)
  - 根据给定正则表达式的匹配拆分此字符串。

字符串与基本数据的相互转化

出于人机交互的目的

- **字符串转换为基本数据类型**
  - Integer 包装类的 public static int **parseInt(String s)** ：可以将由“数字”字符组成的字符串转换为整型。
  - 类似地，使用 java.lang 包中的 Byte 、 Short 、 Long 、 Float 、 Double 类调相应的类方法可以将由“数字”字符组成的字符串，转化为相应的基本数据类型。
- **基本数据类型转换为字符串**
  - 调用 String 类的 public String **valueOf(int n)** 可将 int 型转换为字符串
  - 相应的 valueOf(byte b) 、 valueOf(long l) 、 valueOf(float f) 、 valueOf(double d) 、 valueOf(boolean b) 可由参数的相应类到字符串的转换

字符串与字符、字节数组 (1)

字符串与字符数组

- String 类的构造方法： String(char[]) 和 String(char[] ， int offset ， int length) 分别用字符数组中的全部字符和部分字符创建字符串对象
- String 类提供了将字符串存放到数组中的方法：
  - public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
- 将字符串中的全部字符存放在一个字符数组中的方法：
  - public char[] toCharArray()

字符串与字符、字节数组(2)

各类关系

字符串与字节数组

- String(byte[]) 用指定的字节数组构造一个字符串对象。
  String(byte[] , int offset , int length) 用指定的字节数组的一部分，即从数组起始位置 offset 开始取 length 个字节构造一个字符串对象。
- public byte[] getBytes() 方法使用平台默认的字符编码，将当前字符串转化为一个字节数组。
- public byte[] getBytes(String charsetName) 使用参数指定字符编码，将当前字符串转化为一个字节数组。

```java
public class StringTest_1
{
 //对字符串数组进行排序
 public static void stringSort(String[] arr)
 {
 //采用冒泡排序
 for(int i=0;i<arr.length-1;i++)
 {
  for(int j=0;j<arr.length-1-i;j++)
  {
  //用compareTo方法进行字符串比较
  if(arr[j].compareTo(arr[j+1])>0)
  {
   String temp=arr[j];
   arr[j]=arr[j+1];
   arr[j+1]=temp;
  }
  }
 }
 showArray(arr);
 }
 //定义方法，以[str1,str2,str3]的格式来打印数组
 public static void showArray(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.print(arr[i]+"]\n");
  }
 }
 }
 public static void main(String[] args)
 {
 String arr[]={"nba","abc","cba","zz","qq","haha"};
 //打印数组
 showArray(arr);
 //对数组进行排序并输出
 stringSort(arr);
 }
}
```

字符串练习一

给定一个字符串数组，按照字典顺序，进行大小写排序

思路：

1.对数组排序，可以用选择排序、冒泡排序等等。

2.for循环嵌套，比较，交换位置。

3.不同之处，以前比较的是数字，用的是比较运算符；

现在比较的是字符串对象，应该使用compareTo方法。

排序举例

```
D:\learn>java StringTest_1
[nba,abc,cba,zz,qq,haha]
[abc,cba,haha,nba,qq,zz]
```

排序结果

子串频率统计举例

```java
public class StringTest_2
{
 public static void main(String[] args)
 {
 String str="abcqwabcedcxabcuabcjkabcnmbabc";
 //String str=null;
 try
 {
  int count=countChildStr(str,"abc");
  System.out.println("abc在"+str+"中出现的次数为: "+count);
 }
 catch (NullPointerException ne)
 {
  System.out.println(ne);
 }
 catch(RuntimeException re)
 {
  System.out.println(re);
 }
 }
 public static int countChildStr(String str,String key)
 {
 if(str==null||key==null)
 {
  throw new NullPointerException("空指针异常,源字符串和子串都不能为NULL");
 }
 if(key=="")
 {throw new RuntimeException("调用不合法，子串要有内容");}
 int count=0,index=0;
 while((index=str.indexOf(key,index))!=-1)
 {
  count++;
  index+=key.length();
 }
 return count;
 }
}
```

字符串练习二

一个子串在字符串中出现的次数

思路:

1.用indexOf方法获取子串在字符串中第一次出现的位置index

2.再用indexOf方法获取以(index+子串长度)为起始的剩下的字符串中子串出现的位置，直到字符串中不再包含子串。可用while循环实现。

3.每次找到后用计数器记录即可。

```
D:\learn>java StringTest_2
abc在abcqwabcedcxabcuabcjkabcnmbabc中出现的次数为: 6
```

结果

```java
public class StringTest_3
{
 public static void main(String[] args)
 {
 //创建两个不为空的字符串
 String str1="abxczwsxcvdfas";
 //String str1=null;
 String str2="ghwsxcvxcdbgthnnnrfqwe";
 try
 {
  String str=searchMaxCommonStr(str1,str2);
  System.out.println("最大公共子串是："+str);
 }
 catch (NullPointerException ne)
 {
  System.out.println(ne);
 }
 }
 public static String searchMaxCommonStr(String str1,String str2)
 {
 if(str1==null||str2==null)
  throw new NullPointerException("空指针异常，参数不能为Null");
 //断定较长字符串和较短字符串
 String max=(str1.length()>str2.length())?str1:str2;
 String min=(str1.equals(max))?str2:str1;
 //按长度递减的方式取子串，从min.length~~1
 for(int i=min.length();i>0;i--)
 {
  for(int x=0,y=x+i;y<min.length();x++,y++)
  {
  String childStr=min.substring(x,y);
  //若较长字符串中包含此子串，则找到了
  //否则继续找
  if(max.contains(childStr))
   return childStr;
  }
 }
 return null;
 }
}
```

字符串练习三

找到两个字符串的最大公共子串

最大公共子串

思路：

1.判断较长字符串中是否包含较短字符串，如果包含，则较短字符串则为最大公共子串。

2.如果不包含，就对较短字符串以长度递减的方式取子串，去较长字符串中判断是否包含，如果包含就找到了，不用再找了。

3.重点：对字符串以长度递减的方式取子串

```
D:\learn>java StringTest_3
最大公共子串是：wsxcv
```

结果

字符串练习四

写一个和trim功能相同的方法

去除空格

思路：

1.定义两个变量，用来存储两个角标

2.分别从头和尾遍历字符串，直到找到第一个不为空格的字符

3.截取字符串

```
public class StringTest_4
{
 public static void main(String[] args)
 {
 String str=" abc ws ";
 str=myTrim(str);
 System.out.println(str);
 }
 public static String myTrim(String s)
 {
 int begin=0,end=s.length()-1;
 //从头遍历
 while(begin<=end && s.charAt(begin)==' ')
 {
  begin++;
 }
 //从尾部遍历
 while(begin<=end && s.charAt(end)==' ')
 {
  end--;
 }
 return s.substring(begin,end+1);
 }
}
```

```
D:\learn>java StringTest_4
abc  ws
```

分割子串举例

JavaStringSplitEmp.java

```java
public class JavaStringSplitEmp {

    public static void main(String args[]){
        String str = "www-runoob-com";
        String[] temp;
        String delimeter = "-";  // 指定分割字符
        temp = str.split(delimeter); // 分割字符串

        // 普通 for 循环
        for(int i =0; i < temp.length ; i++){
            System.out.println(temp[i]);
            System.out.println("");
        }

        System.out.println("------java for each循环输出的方法-----");
        String str1 = "www.runoob.com";
        String[] temp1;
        String delimeter1 = "\\.";  // 指定分割字符，. 号需要转义
        temp1 = str1.split(delimeter1); // 分割字符串
        for(String x : temp1){
            System.out.println(x);
            System.out.println("");
        }
    }
}
```

Console

```
<terminated> JavaStringSplitEmp [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\ja
www

runoob

com

------java for each循环输出的方法-----
www

runoob

com
```

java.lang.string.split

split 方法

将一个字符串分割为子字符串，然后将结果作为字符串数组返回。

\\会转义成反斜杠，反斜杠本身就是转义符，所有就成了“\.”，在进行转义就是.，所以\\.实际上是“.”。

作业

忽略

1. 将一个字符串进行反转。将字符串中指定部分进行反转。比如将“ abcdefg ”反转为” abfedcg ”
2. 获取一个字符串在另一个字符串中出现的次数。
   比如：获取“ ab ”在 “ abkkcadkabkebfkabkskab ”中出现的次数
3. 获取两个字符串中最大相同子串。比如：
   str1 = "abcwerthelloyuiodef“;str2 = "cvhellobnm"
   提示：将短的那个串进行长度依次递减的子串与较长的串比较。
4. 对字符串中字符进行自然顺序排序。

提示：

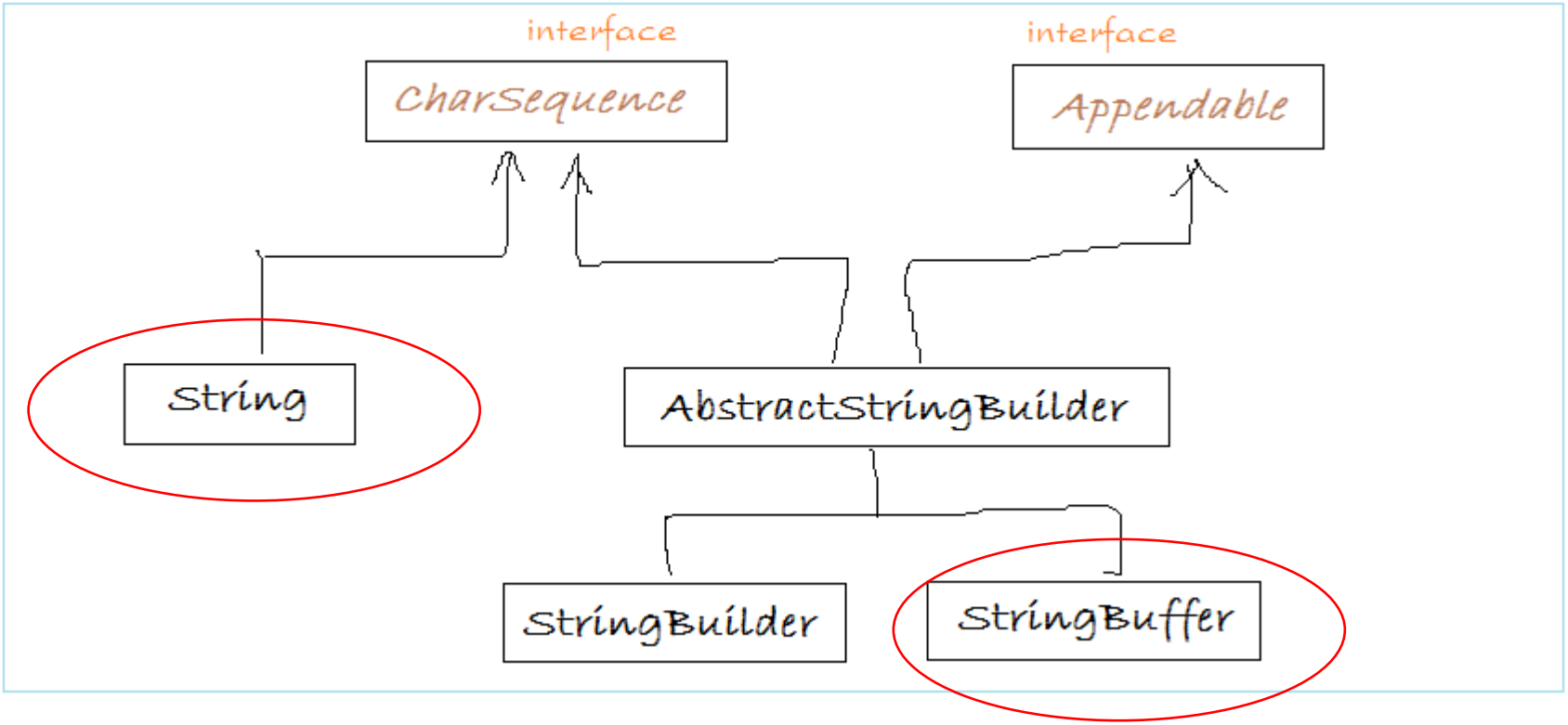
1 ）字符串变成字符数组。

2 ）对数组排序，选择，冒泡， Arrays.sort();

3 ）将排序后的数组变成字符串。

StringBuffer 类

- java.lang.StringBuffer 代表**可变的字符序列**，可以对字符串内容进行增删。
- 很多方法与 String 相同，但 StingBuffer 是可变长度的。
- StringBuffer 是一个容器。



StringBuffer 是可变的。它可以在长度和内容方面发生变化。 StringBuffer 是线程安全的，这意味着它们已经同步方法来控制访问，以便只有一个线程可以在同一时间访问一个 StringBuffer 对象同步代码。因此，StringBuffer 的对象通常在多线程环境中是安全的，使用多个线程可以试图同时访问相同 StringBuffer 对象。

StringBuffer 类

- StringBuffer 类有三个构造方法：

1．StringBuffer() 初始容量为 16 的字符串缓冲区

2．StringBuffer(int size) 构造指定容量的字符串缓冲区

3．StringBuffer(String str) 将内容初始化为指定字符串内容

StringBuffer 类

- String s = new String(" 我喜欢学习 ");
- StringBuffer buffer = new StringBuffer(“我喜欢学习”);
  - buffer.append(" 数学 ");

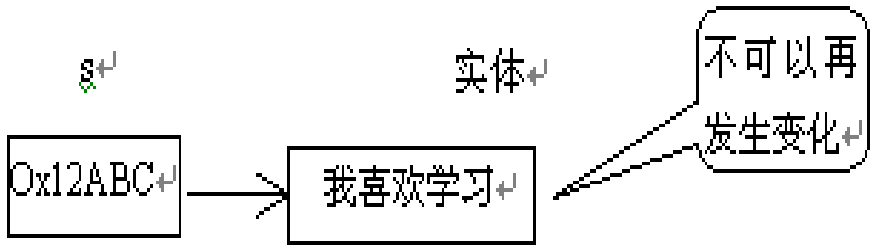


图 9.9 实体不可变

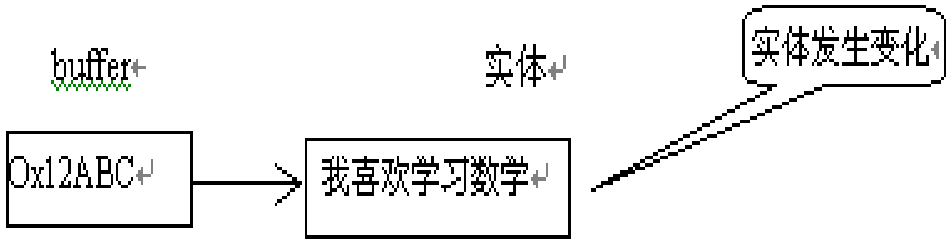
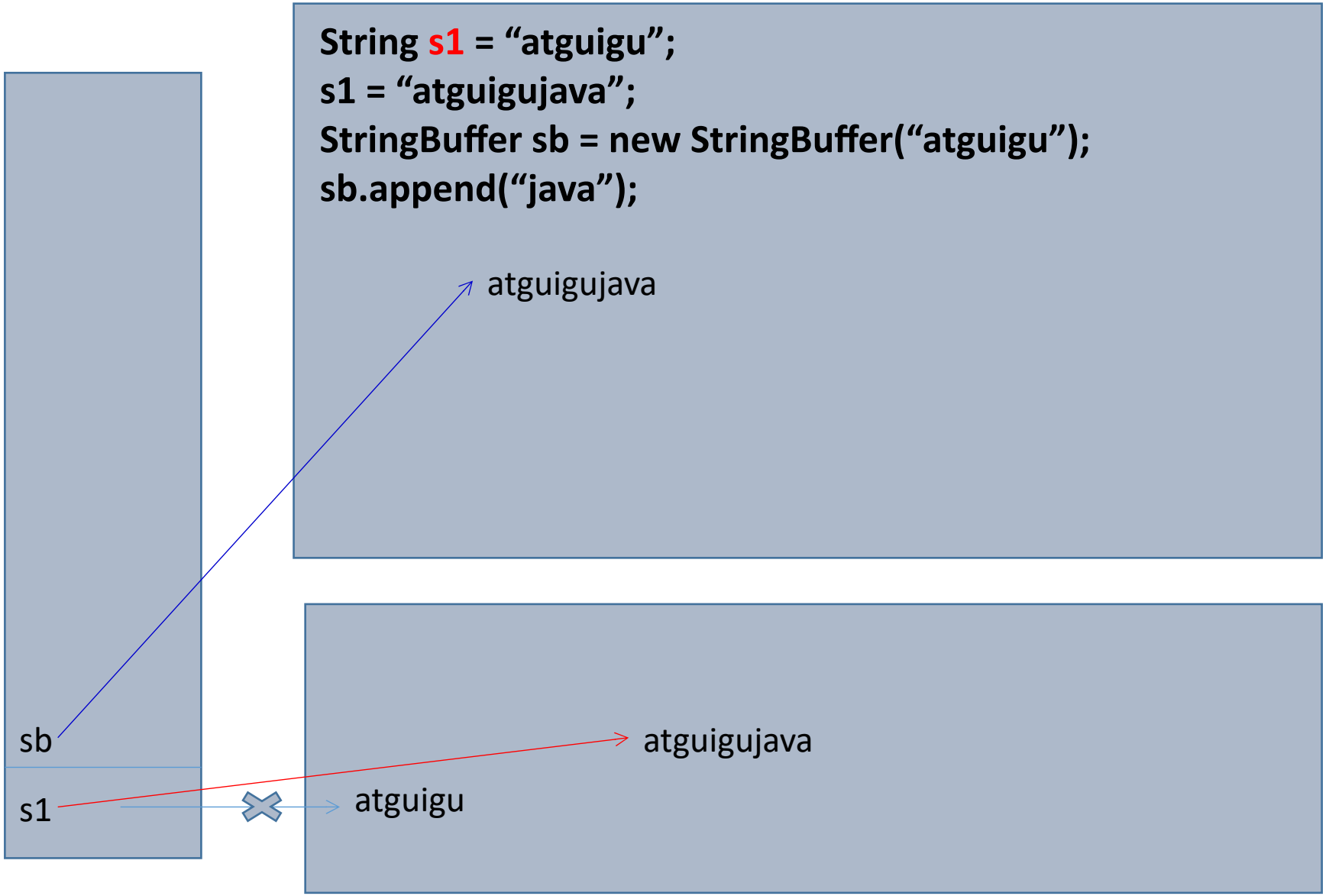


图 9.10 实体可变

区别


String s1 = “atguigu”;
s1 = “atguigujava”;
StringBuffer sb = new StringBuffer(“atguigu”);
sb.append(“java”);
atguigujava
sb
s1
atguigujava
atguigu

可变字符的处理
了解即可

StringBuffer 类的常用方法

StringBuffer **append**(String s), StringBuffer append(int n) ,

StringBuffer append(Object o) , StringBuffer append(char n),

StringBuffer append(long n), StringBuffer append(boolean n),

StringBuffer **insert**(int index, String str)

public StringBuffer **reverse**()

StringBuffer **delete**(int startIndex, int endIndex)

public char **charAt**(int n)

public void **setCharAt**(int n ,char ch)

StringBuffer **replace**(int startIndex ,int endIndex, String str)

public int **indexOf**(String str)

public String **substring**(int start,int end)

public int **length**()

可变字符的处理
了解即可

StringBuilder 类

- **StringBuilder 和 StringBuffer 非常类似，均代表可变的字符序列，而且方法也一样**

➢String ：不可变字符序列

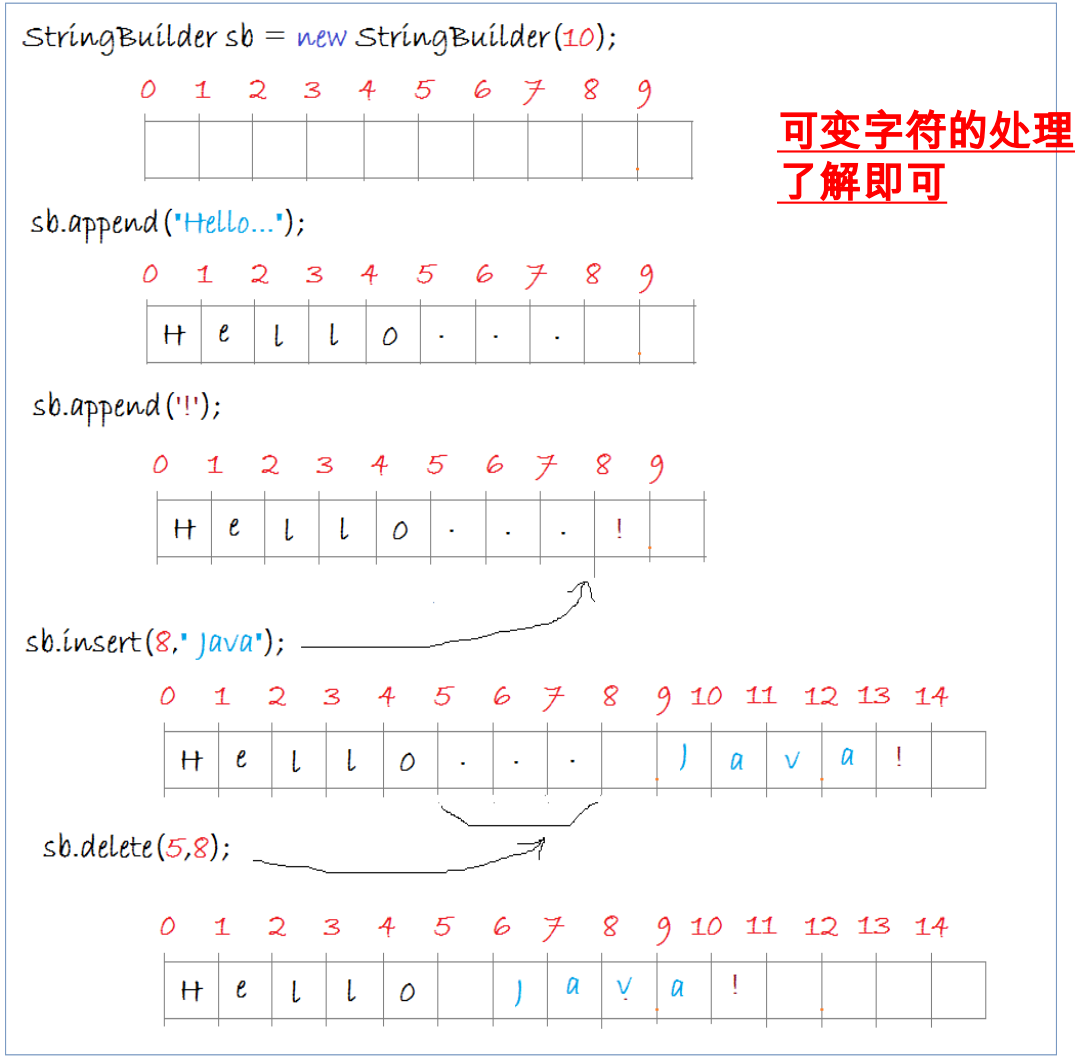
➢StringBuffer ：可变字符序列、效率低、线程安全

➢StringBuilder(JDK1.5) ：可变字符序列、效率高、线程不安全

StringBuilder 类非常相似的 StringBuffer ，不同之处在于它的访问不同步的，因此，它不是线程安全的。由于不同步， StringBuilder 的性能可以比 StringBuffer 更好。因此，如果在单线程环境中工作，使用 StringBuilder ，而不是 StringBuffer 可能会有更高的性能。这也类似其他情况，如 StringBuilder 的局部变量 (即一个方法中的一个变量) ，其中只有一个线程会访问一个 StringBuilder 对象。

StringBuilderDemo.java

```java
public class StringBuilderDemo {

    public static void main(String[] args) {

        // Create StringBuilder object
        // with no characters in it and
        // an initial capacity specified by the capacity argument
        StringBuilder sb = new StringBuilder(10);

        // Append the string Hello ... on sb.
        sb.append("Hello...");
        System.out.println("- sb after appends a string: " + sb);

        // append a character
        char c = '!';
        sb.append(c);
        System.out.println("- sb after appending a char: " + sb);

        // Insert a string at index 5
        sb.insert(8, " Java");
        System.out.println("- sb after insert string: " + sb);

    }

    // Delete substring at index 5 to 8
    sb.delete(5,8);

    System.out.println("- sb after delete: " + sb);
    }
}
```



可变字符的处理
了解即可

Console

<terminated> StringBuilderDemo [Java Application

```
- sb after appends a string: Hello...
- sb after appending a char: Hello...!
- sb after insert string: Hello... Java!
- sb after delete: Hello Java!
```

可变字符的处理
了解即可
效率举例

三者的效率测试

```
String text = "";
long startTime = 0L;
long endTime = 0L;
StringBuffer buffer = new StringBuffer("");
StringBuilder builder = new StringBuilder("");
startTime = System.currentTimeMillis();
for(int i = 0;i<20000;i++){
buffer.append(String.valueOf(i));}
endTime = System.currentTimeMillis();
System.out.println("StringBuffer 的执行时间："+(endTime-startTime));
startTime = System.currentTimeMillis();
for(int i = 0;i<20000;i++){
builder.append(String.valueOf(i));}
endTime = System.currentTimeMillis();
System.out.println("StringBuilder 的执行时间："+(endTime-startTime));
startTime = System.currentTimeMillis();
for(int i = 0;i<20000;i++){
text = text + i;}
endTime = System.currentTimeMillis();
System.out.println("String 的执行时间："+(endTime-startTime));
```

*Testing.java

```java
public class Testing {
    public static void main(String[] args){
        String text = "";
        long startTime = 0L;
        long endTime = 0L;
        StringBuffer buffer = new StringBuffer("");
        StringBuilder builder = new StringBuilder("");
        startTime = System.currentTimeMillis();
        for(int i = 0;i<20000;i++){
        buffer.append(String.valueOf(i));}
        endTime = System.currentTimeMillis();
        System.out.println("StringBuffer的执行时间："+(endTime-startTime));
        startTime = System.currentTimeMillis();
        for(int i = 0;i<20000;i++){
        builder.append(String.valueOf(i));}
        endTime = System.currentTimeMillis();
        System.out.println("StringBuilder的执行时间："+(endTime-startTime));
        startTime = System.currentTimeMillis();
        for(int i = 0;i<20000;i++){
        text = text + i;}
        endTime = System.currentTimeMillis();
        System.out.println("String的执行时间："+(endTime-startTime));
    }
}
```

可变字符的处理
了解即可
效率举例

Console

<terminated> Testing [Java App

```
StringBuffer的执行时间：8
StringBuilder的执行时间：6
String的执行时间：997
```

二、日期类

1.java.lang.System 类

System 类提供的 public static long currentTimeMillis()
用来返回当前时间与 1970 年 1 月 1 日 0 时 0 分 0 秒
之间以毫秒为单位的时间差。

- **此方法适于计算时间差。**

- 计算世界时间的主要标准有：
  - UTC(Universal Time Coordinated)
  - GMT(Greenwich Mean Time)
  - CST(Central Standard Time)

日期类

2. java.util.Date 类

表示特定的瞬间，精确到毫秒

- **构造方法：**
  - **Date( ) 使用 Date 类的无参数构造方法创建的对象可以获取本地当前时间。**
  - **Date(long date)**
- **常用方法**
  - **getTime():** 返回自 1970 年 1 月 1 日 00:00:00 GMT 以来此 Date 对象表示的毫秒数。
  - **toString():** 把此 Date 对象转换为以下形式的 String ：dow mon dd hh:mm:ss zzz yyyy 其中： dow 是一周中的某一天 (Sun, Mon, Tue, Wed, Thu, Fri, Sat) ， zzz 是时间标准。

日期类
了解

```
import java.util.Date;

public void testDate(){
        Date date = new Date();
        System.out.println(date);
        System.out.println(System.currentTimeMillis());
        System.out.println(date.getTime());
        Date date1 = new Date(date.getTime());
        System.out.println(date1.getTime());
        System.out.println(date1.toString());
}
```

日期类
了解

Date 类的 API 不易于国际化，大部分被废弃了，

java.text.Simp
leDateFormat 类是一个不与语言环境有关的方式来格式化和解析日期的具体类。

- 它允许进行**格式化（日期◊文本）**、**解析（文本◊日期）**

◆格式化：

- **SimpleDateFormat()** ：默认的模式和语言环境创建对象
- **public SimpleDateFormat(String pattern)** ：该构造方法可以用参数 pattern 指定的格式创建一个对象，该对象调用：
- **public String format(Date date)** ：方法格式化时间对象 date

◆解析：

- **public Date parse(String source)** ：从给定字符串的开始解析文本，以生成一个日期。

日期类
了解

```
Date date = new Date();  // 产生一个 Date 实例
// 产生一个 formater 格式化的实例
SimpleDateFormat formater = new SimpleDateFormat();
        System.out.println(formater.format(date));// 打印输出默认的格式
        SimpleDateFormat formater2 = new SimpleDateFormat(
                                        "yyyy 年 MM 月 dd 日  EEE HH:mm:ss");
        System.out.println(formater2.format(date));
        // 实例化一个指定的格式对象
        // 按指定的格式输出
        try {
                Date date2 = formater2.parse( “ 2008 年 08 月 08 日 星期一
08:08:08");
                 // 将指定的日期解析后格式化按指定的格式输出
                System.out.println(date2.toString());
        } catch (ParseException e) {
                e.printStackTrace();
        }
```

日期类

3. java.util.Calendar(日历) 类

Calendar 是一个抽象基类，主用用于完成日期字段之间相互操作的功能。

- 获取 Calendar 实例的方法
  - 使用 Calendar.getInstance() 方法
  - 调用它的子类 GregorianCalendar 的构造器。

- 一个 Calendar 的实例是系统时间的抽象表示，通过 get(int field) 方法来取得想要的时间信息。比如 YEAR 、 MONTH 、 DAY_OF_WEEK 、 HOUR_OF_DAY 、 MINUTE 、 SECOND
  - public void set(int field,int value)
  - public void add(int field,int amount)
  - public final Date getTime()
  - public final void setTime(Date date)

日期类
了解

```
Calendar calendar = Calendar.getInstance();
// 从一个 Calendar 对象中获取 Date 对象
Date date = calendar.getTime();
// 使用给定的 Date 设置此 Calendar 的时间
calendar.setTime(date);
calendar.set(Calendar.DAY_OF_MONTH, 8);
System.out.println(" 当前时间日设置为 8 后 , 时间是 :" +calendar.getTime());
calendar.add(Calendar.HOUR, 2);
System.out.println(" 当前时间加 2 小时后 , 时间是 :" +calendar.getTime());
calendar.add(Calendar.MONTH, -2);
System.out.println(" 当前日期减 2 个月后 , 时间是 :" +calendar.getTime());
```

CalendarDemo.java

```java
import java.util.Calendar;

public class CalendarDemo {

    public static void main(String[] args) {

        // create a calendar
        Calendar cal = Calendar.getInstance();

        // print current date
        System.out.println("The current date is : " + cal.getTime());

        // add 20 days to the calendar
        cal.add(Calendar.DATE, 20);
        System.out.println("20 days later: " + cal.getTime());

        // subtract 2 months from the calendar
        cal.add(Calendar.MONTH, -2);
        System.out.println("2 months ago: " + cal.getTime());

        // subtract 5 year from the calendar
        cal.add(Calendar.YEAR, -5);
        System.out.println("5 years ago: " + cal.getTime());
    }
}
```

Console

<terminated> CalendarDemo [Java Application] C:\Program File

```
The current date is : Fri Nov 30 08:44:11 CST 2018
20 days later: Thu Dec 20 08:44:11 CST 2018
2 months ago: Sat Oct 20 08:44:11 CST 2018
5 years ago: Sun Oct 20 08:44:11 CST 2013
```

日期类
举例

数学类

三、 Math 类

java.lang.Math 提供了一系列静态方法用于科学计算；其方法的参数和返回值类型一般为 double 型。

abs　绝对值

acos,asin,atan,cos,sin,tan　三角函数

sqrt　平方根

pow(double a,doble b)　a 的 b 次幂

log　自然对数

exp　e 为底指数

max(double a,double b)

min(double a,double b)

random()　返回 0.0 到 1.0 的随机数

long round(double a)　double 型数据 a 转换为 long 型（四舍五入）

toDegrees(double angrad)　弧度—>角度

toRadians(double angdeg)　角度—>弧度

数学类
举例

StringTest_1.java | TestMath.java

```
public class TestMath {
    public static void main (String []args)
    {
        System.out.println("90 度的正弦值：" + Math.sin(Math.PI/2));
        System.out.println("0度的余弦值：" + Math.cos(0));
        System.out.println("60度的正切值：" + Math.tan(Math.PI/3));
        System.out.println("1的反正切值： " + Math.atan(1));
        System.out.println("π/2的角度值：" + Math.toDegrees(Math.PI/2));
        System.out.println(Math.PI);
    }
}
```

Console

```
<terminated> TestMath [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2
90 度的正弦值：1.0
0度的余弦值：1.0
60度的正切值：1.7320508075688767
1的反正切值： 0.7853981633974483
π/2的角度值：90.0
3.141592653589793
```

BigInteger 类了解

四、 BigInteger 类

Integer类作为int的包装类，能存储的最大整型值为$2^{31}-1$，BigInteger类的数字范围较Integer类的数字范围要大得多，可以支持任意精度的整数。

- 构造器
  - **BigInteger**(String val)
- 常用方法
  - public BigInteger **abs**()
  - public BigInteger **add**(BigInteger val)
  - public BigInteger **subtract**(BigInteger val)
  - public BigInteger **multiply**(BigInteger val)
  - public BigInteger **divide**(BigInteger val)
  - public BigInteger **remainder**(BigInteger val)
  - public BigInteger **pow**(int exponent)
  - public BigInteger[] **divideAndRemainder**(BigInteger val)

BigInteger 类了解

四、 BigDecimal 类

可以使用 **BigInteger** 操作大整数；可以使用 **BigDecimal** 指定小数的保留位数

如果在操作的时候一个整型数据已经超过了整数的最大类型长度 long 的话，则此数据就无法装入，所以，此时要使用 BigInteger 类进行操作。

No.	方法	类型	描述
1	public BigInteger(String val)	构造	将一个字符串变为BigInteger类型的数据
2	public BigInteger add(BigInteger val)	普通	加法
3	public BigInteger subtract(BigInteger val)	普通	减法
4	public BigInteger multiply(BigInteger val)	普通	乘法
5	public BigInteger divide(BigInteger val)	普通	除法
6	public BigInteger max(BigInteger val)	普通	返回两个大数字中的最大值
7	public BigInteger min(BigInteger val)	普通	返回两个大数字中的最小值
8	public BigInteger[] divideAndRemainder(BigInteger val)	普通	除法操作，数组的第一个元素为除法的商，第二个元素为除法的余数

StringTest_1.java　TestMath.java　*BigIntegerDemo1.java

```java
import java.math.BigInteger;

public class BigIntegerDemo1 {
    public static void main(String[] args) {
        BigInteger bi1 = new BigInteger("123456789"); // 声明BigInteger对象
        BigInteger bi2 = new BigInteger("987654321"); // 声明BigInteger对象
        System.out.println("加法操作：" + bi2.add(bi1)); // 加法操作
        System.out.println("减法操作：" + bi2.subtract(bi1)); // 减法操作
        System.out.println("乘法操作：" + bi2.multiply(bi1)); // 乘法操作
        System.out.println("除法操作：" + bi2.divide(bi1));// 除法操作
        System.out.println("最大数：" + bi2.max(bi1)); // 求出最大数
        System.out.println("最小数：" + bi2.min(bi1)); // 求出最小数
        BigInteger result[] = bi2.divideAndRemainder(bi1);// 求出余数的除法操作
        System.out.println("商是：" + result[0] + "；余数是：" + result[1]);
    }
}
```

Console

```
<terminated> BigIntegerDemo1 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (20
加法操作：1111111110
减法操作：864197532
乘法操作：121932631112635269
除法操作：8
最大数：987654321
最小数：123456789
商是：8；余数是：9
```

BigInteger 类
了解
举例

一般的 Float 类和 Double 类可以用来做科学计算或工程计算，但在**商业计算中，要求数字精度比较高，故用到 java.math.BigDecimal 类**。 BigDecimal 类支持任何精度的定点数。

BigDecimal 类了解

No.	方法	类型	描述
1	public BigDecimal(double val)	构造	将double表示形式转换为BigDecimal
2	public BigDecimal(int val)	构造	将int表示形式转换为BigDecimal
3	public BigDecimal(String val)	构造	将字符串表示形式转换为BigDecimal
4	public BigDecimal add(BigDecimal augend)	普通	加法
5	public BigDecimal subtract(BigDecimal subtrahend)	普通	减法
6	public BigDecimal multiply(BigDecimal multiplicand)	普通	乘法
7	public BigDecimal divide(BigDecimal divisor)	普通	除法

*BigDecimalDemo01.java

```java
import java.math.BigDecimal;

public class BigDecimalDemo01 {
    public static void main(String[] args) {
        System.out.println("加法运算：" + MyMath.round(MyMath.add(10.345,3.333),1));
        System.out.println("减法运算：" + MyMath.round(MyMath.sub(10.345,3.333),3));
        System.out.println("乘法运算：" + MyMath.round(MyMath.mul(10.345,3.333),4));
        System.out.println("除法运算：" + MyMath.div(10.345,3.333,3));
    }
}
class MyMath{
    public static double add(double d1,double d2){    //进行加法计算
        BigDecimal b1 = new BigDecimal(d1);
        BigDecimal b2 = new BigDecimal(d2);
        return b1.add(b2).doubleValue();
    }
    public static double sub(double d1,double d2){    //进行减法计算
        BigDecimal b1 = new BigDecimal(d1);
        BigDecimal b2 = new BigDecimal(d2);
        return b1.subtract(b2).doubleValue();
    }
    public static double mul(double d1,double d2){    //进行乘法计算
        BigDecimal b1 = new BigDecimal(d1);
        BigDecimal b2 = new BigDecimal(d2);
        return b1.multiply(b2).doubleValue();
    }
    public static double div(double d1,double d2,int len){    //进行除法计算
        BigDecimal b1 = new BigDecimal(d1);
        BigDecimal b2 = new BigDecimal(d2);
        return b1.divide(b2,len,BigDecimal.ROUND_HALF_UP).doubleValue();
    }
    public static double round(double d,int len){   //进行四舍五入
        BigDecimal b1 = new BigDecimal(d);
        BigDecimal b2 = new BigDecimal(1); //技巧
        return b1.divide(b2,len,BigDecimal.ROUND_HALF_UP).doubleValue();
    }
}
```

Console

```
<terminated> BigDecimalDemo01 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2018
加法运算：13.7
减法运算：7.012
乘法运算：34.4799
除法运算：3.104
```

BigDecimal 类
了解
举例