**3.1、String类（重点）**

**3.1.1、String类两种对象实例化方式**

对于String在之前已经学习过了基本使用，就是表示字符串，那么当时使用的形式**采用了直接赋值**：

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "Hello" ; // 定义字符串  System.out.println(str) ;  }  } |

对于String而言肯定是一个类，那么程序之中出现的str应该就是这个类的对象，那么就证明以上的赋值操作实际上就表示要为String类的对象进行实例化操作。

但String毕竟是一个类，那么类之中一定会存在构造方法，**String类的构造：public String(String str)；**

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = new String("Hello") ; // 定义字符串  System.out.println(str) ;  }  } |

发现现在也可以通过构造方法为String类对象实例化。

**3.1.2、字符串比较**

如果说现在有两个int型变量，如果要想知道是否相等，使用“==”进行验证。

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  int x = 10 ;  int y = 10 ;  System.out.println(x == y) ;  }  } |

但是，同样的操作，现在换成String完成。

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str1 = "Hello" ;  String str2 = new String("Hello") ;  String str3 = str2 ; // 引用传递  System.out.println(str1 == str2) ; // false  System.out.println(str1 == str3) ; // false  System.out.println(str2 == str3) ; // true  }  } |

发现使用“==”好象最终的判断结果完全是不一样的，那么下面通过内存关系图完成。

现在使用了“==”的确是完成了相等的判断，但是最终判断的是两个对象（现在的对象是字符串）判断是否相等，属于数值判断 —— 判断的是两个对象的内存地址数值，并没有判断内容，而要想完成字符串内容的判断，则就必须使用到String类的操作方法：public boolean equals(String str)（将方法暂时变了）。

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str1 = "Hello" ;  String str2 = new String("Hello") ;  String str3 = str2 ; // 引用传递  System.out.println(str1.equals(str2)) ; // true  System.out.println(str1.equals(str3)) ; // true  System.out.println(str2.equals(str3)) ; // true  }  } |

现在比较的不再是内存地址的数值，而是两个字符串的内容。

**面试题：**请解释字符串比较之中“==”和equals()的区别？

· ==：比较的是两个字符串内存地址的数值是否相等，属于数值比较；

· equals()：比较的是两个字符串的内容，属于内容比较。

**以后进行字符串相等判断的时候都使用equals()。**

**3.1.3、字符串常量是String的匿名对象**

如果在程序之中定义了字符串（使用“"”），那么这个就表示一个String对象，因为在各个语言之中没有关于字符串数据类型定义，而Java将其简单处理了，所以感觉上存在了字符串类型。

**范例：**验证字符串是对象的概念

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "Hello" ;  // 通过字符串调用方法  System.out.println(**"Hello"**.equals(str)) ;  }  } |

匿名对象可以调用类之中的方法与属性，以上的字符串可以调用了equals()方法，那么一定是一个对象。

**小技巧：**关于字符串与字符串常量的判断

例如：在实际工作之中会有这样一操作，要求用户输入一个内容，之后判断此内容是否与指定的内容相同。

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "Hello" ; // 假设是输入的  if (str.equals("Hello")) {  System.out.println("条件满足。") ;  }  }  } |

但，既然数据由用户自己输入，那么就有可能没有输入的内容。

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = **null** ; // 假设是输入的  if (str.equals("Hello")) {  System.out.println("条件满足。") ;  }  }  } |
| Exception in thread "main" java.lang.**NullPointerException**  at StringDemo.main(StringDemo.java:4) |

而如果说现在将代码反过来操作：

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = null ; // 假设是输入的  if ("Hello".equals(str)) {  System.out.println("条件满足。") ;  }  }} |

因为字符串常量就是一个匿名对象，匿名对象永远不可能为null，这样的比较才更加的合理。

**3.1.4、String类的两种实例化方式的区别**

对于String类的对象存在了两种实例化的操作形式，那么这两种有什么区别，在开发之中应该使用那一种更好呢？

**1、 分析直接赋值的情况：**

|  |
| --- |
| String str = "Hello" ; |

发现现在只开辟了一块堆内存空间和一块栈内存空间。那么随后将代码进行一些修改。

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str1 = "Hello" ;  String str2 = "Hello" ;  String str3 = "Hello" ;  System.out.println(str1 == str2) ; // true  System.out.println(str1 == str3) ; // true  System.out.println(str2 == str3) ; // true  }  } |

**解释：**关于直接赋值操作之中，字符串比较都是相同的原因

在String类进行设计的时候采用了一种称为共享设计模式的概念，在每一个运行的JVM底层存在一个字符串的对象池（Object Pool），如果用户采用了直接赋值的方式，会将字符串的内容放入到池之中，以供其他继续使用直接赋值方式的String对象使用，如果新声明的字符串内容不再池之中，则会开辟一个新的，继续放到池，以供下次使用。

**2、 分析构造方法赋值的情况：**

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = new String("Hello") ;  System.out.println(str) ;  }  } |

使用构造方法的方式开辟的字符串对象，实际上会开辟两块空间，其中有一块空间将称为垃圾。

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str1 = new String("Hello") ;  String str2 = "Hello" ; // 入池  String str3 = "Hello" ; // 使用池对象  System.out.println(str1 == str2) ; // false  System.out.println(str1 == str3) ; // false  System.out.println(str2 == str3) ; // true  }  } |

通过上面的程序可以发现，使用构造方法实例化的String对象，不会入池，所以，只能自己使用。可是在String类之中为了方便操作提供了一种称为手工入池的方法：**public String intern()**。

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str1 = new String("Hello").**intern() ;**  String str2 = "Hello" ; // 入池  String str3 = "Hello" ; // 使用池对象  System.out.println(str1 == str2) ; // true  System.out.println(str1 == str3) ; // true  System.out.println(str2 == str3) ; // true  }  } |

**面试题：**请解释String类的两种对象实例化方式的区别？

· 直接赋值：只开辟一块堆内存空间，字符串的内容可以自动入池，以供下次使用；

· 构造方法：开辟两块堆内存空间，有一块将成为垃圾，并且不能自动入池，使用intern()手工入池。

在日后的所有开发之中，String对象的实例化永远都采用直接赋值的方式完成。

**3.1.5、字符串的内容一旦声明则不可改变**

字符串类的操作特点决定：字符串不可能去修改里面的内容，所以，如果以后在开发之中有如下程序。

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "Hello " ;  str += "World " ;  str = str + "!!!" ;  System.out.println(str) ;  }  } |

通过以上的代码可以发现，字符串内容的更改，实际上改变的是字符串对象的引用过程，并且会伴随有大量的垃圾出现，那么对于以下的代码实际之中应该避免：

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "" ;  for (int x = 0 ; x < 1000 ; x ++) {  str += x ;  }  System.out.println(str) ;  }  } |

但是这种代码需要“断开-连接”String对象1000次，会产生大量垃圾，所以不能够去使用。

**3.2、String类的常用方法（重点，背）**

String类除了之前所介绍的两个方法之外，还存在着大量的其他操作，考虑到String类在实际的工作之中使用非常的广泛，那么就建议大家都背下来，以下所讲解的每一个方法：**记住方法的名称、返回值类型、参数的类型及个数、方法的作用**。

以后所有的开发都要和文档挂勾，而且文档不许使中文，因为在所有的Java学习之中，有十几份文档呢。文档之中会对每一个类进行详细的解释，而一般的解释类的文档的结构：

· 类的基本定义；

· 类的说明；

· 类的主体部分：

|- 类之中的成员（Field）；

|- 类中的构造方法（Constructor）；

|- 类中的普通方法（Method）。

· 对每一个成员、构造方法、普通方法的作用进行详细说明，包括参数的作用。

**3.2.1、字符串与字符**

在很多的语言之中，都强调，字符串由字符数组所组成，那么这一概念在Java的String类之中也有体现。

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **方法名称** | **类型** | **描述** |
| 1 | public String(char[] value) | 构造 | 将全部的字符数组内容变为字符串 |
| 2 | public String(char[] value, int offset, int count) | 构造 | 将部分字符数组变为字符串，offset表示开始点，count表示要操作的长度 |
| 3 | public char charAt(int index) | 普通 | 取得指定索引位置上的字符 |
| 4 | public char[] toCharArray() | 普通 | 将字符串转换为字符数组 |

**范例：**验证charAt()方法，这个方法是一个标志性方法

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "helloworld" ;  char c = str.charAt(7) ;  System.out.println(c) ;  }  } |

**范例：**字符串和字符数组转换，完成一个小写字符串变为大写字符串的操作，小写字母和大写字母差了32

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "helloworld" ;  char data [] = str.toCharArray() ; // 字符串变为字符数组  for (int x = 0 ; x < data.length ; x ++) {  System.out.print(data[x] + "、") ;  data [x] -= 32 ; // 变大写  }  System.out.println() ;  System.out.println("全部字符数组变为字符串：" + new String(data)) ;  System.out.println("部分字符数组变为字符串：" + new String(data,0,5)) ;  }  } |

**思考题：**现在要求判断一个字符串是否由数字所组成

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  char c = '8' ;  System.out.println(c >= '0' && c <= '9') ;  System.out.println((int) c) ;  }  } |

**思路：**将一个字符串首先变为字符数组，而后依次判断字符数组之中的每一个字符是否是数字，如果全是，则返回true，否则返回false。

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = " 1a 23" ;  System.out.println(isNumber(str)) ;  }  **public static boolean isNumber(String temp) {**  **char data [] = temp.toCharArray() ; // 变为字符数组**  **for (int x = 0 ; x < data.length ; x ++) {**  **if (data[x] < '0' || data[x] > '9') {**  **return false ; // 不是数字**  **}**  **}**  **return true ;**  **}**  } |

以上的这些操作只是给了一个基本的思路，实际上不可能如此完成的，但是这些基本的程序逻辑应该明白。

**3.2.2、字符串与字节**

字符串除了与字符可以互相转换之外，还可以和字节互相转换，操作方法：

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **方法名称** | **类型** | **描述** |
| 1 | public String(byte[] bytes) | 构造 | 将全部的字节数组变为字符串 |
| 2 | public String(byte[] bytes, int offset, int length) | 构造 | 将部分的字节数组变为字符串 |
| 3 | public byte[] getBytes() | 普通 | 将字符串变为字节数组 |
| 4 | public byte[] getBytes(String charsetName) throws UnsupportedEncodingException | 普通 | 字符串转码操作 |

**范例：**完成一个小写字母变为大写字母的操作

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "helloworld" ;  byte data [] = str.getBytes() ; // 字符串变为字节数组  for (int x = 0 ; x < data.length ; x ++) {  System.out.print(data[x] + "、") ;  data [x] -= 32 ; // 变大写  }  System.out.println() ;  System.out.println("全部字节数组变为字符串：" + new String(data)) ;  System.out.println("部分字节数组变为字符串：" + new String(data,0,5)) ;  }  } |

一般情况下，在程序之中如果要想操作字节数组只有两种情况：

· **情况一：**需要进行编码的转换时；

· **情况二：**数据要进行传输的时候。

**3.2.3、字符串比较**

之前学习了equals()用于比较两个字符串的内容是否相同。

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **方法名称** | **类型** | **描述** |
| 1 | public boolean equals(String anObject) | 普通 | 区分大小写的相等判断 |
| 2 | public boolean equalsIgnoreCase(String anotherString) | 普通 | 不区分大小写比较是否相等 |
| 3 | public int compareTo(String anotherString) | 普通 | 比较两个字符串的大小 |

**范例：**equals()方法

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str1 = "helloworld" ;  String str2 = "HELLOWORLD" ;  System.out.println(str1.equals(str2)) ;  System.out.println(str1.equalsIgnoreCase(str2)) ;  }  } |

如果现在要想比较两个字符串的大小关系，那么就必须使用compareTo()方法完成，而这个方法返回int型数据，而**这个int型数据有三种结果：大于（返回结果大于0）、小于（返回结果小于0）、等于（返回结果为0）**。

**范例：**比较大小

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str1 = "Helloworld" ;  String str2 = "HELLOWORLD" ;  System.out.println(str1.compareTo(str2)) ;  System.out.println("Hello".compareTo("Hello")) ;  }  } |

对于compareTo()方法一定要记住其返回的类型，以后还会有其他的深入讲解。

**3.2.4、字符串查找**

在Oracle之中如果要想查找一个子字符串是否在指定的字符串之中存在的函数使用的是inst()，而这样的功能在String类之中也是有所提供的，而且提供的方法还有很多种。

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **方法名称** | **类型** | **描述** |
| 1 | public boolean contains(String s) | 普通 | 查找指定的子字符串是否存在，JDK 1.5之后有 |
| 2 | public int indexOf(String str) | 普通 | 从头查找指定字符串的位置，找不到返回-1 |
| 3 | public int indexOf(String str, int fromIndex) | 普通 | 由指定位置向后查找字符串的位置，找不到返回-1 |
| 4 | public int lastIndexOf(String str) | 普通 | 由后向前查找字符串的位置，找不到返回-1 |
| 5 | public int lastIndexOf(String str, int fromIndex) | 普通 | 从指定位置由后向前查找 |
| 6 | public boolean startsWith(String prefix) | 普通 | 判断是否以指定的字符串开头 |
| 7 | public boolean startsWith(String prefix, int toffset) | 普通 | 从指定位置判断是否以指定字符串开头，JDK 1.7 |
| 8 | public boolean endsWith(String suffix) | 普通 | 判断是否以指定的字符串结尾 |

**范例：**判断开头和结尾操作

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "\*\*@@hello##" ;  System.out.println(str.startsWith("\*\*")) ;  System.out.println(str.startsWith("@@",2)) ;  System.out.println(str.endsWith("##")) ;  }  } |

**范例：**使用contains()方法查找字符串是否存在，直接返回boolean，用于各种的执行判断

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "helloworld" ;  System.out.println(str.contains("hello")) ;  System.out.println(str.contains("xx")) ;  }  } |

**范例：**古老的操作，indexOf()

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "helloworld" ;  if (str.indexOf("hello") != -1) {  System.out.println("字符串存在：" + str.indexOf("hello")) ;  }  System.out.println(str.indexOf("xxx")) ; // -1  System.out.println(str.indexOf("l",5)) ;  System.out.println(str.lastIndexOf("l")) ;  }  } |

在一些古老的教材上，出现的字符串查找都属于indexOf()方法，但是这个方法没有contains()好用。

**3.2.5、字符串替换操作**

Oracle中的替换函数是replace()，那么String类之中的替换操作有如下几个方法：

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **方法名称** | **类型** | **描述** |
| 1 | public String replaceAll(String regex, String replacement) | 普通 | 全部替换 |
| 2 | public String replaceFirst(String regex, String replacement) | 普通 | 替换首个 |

**范例：**验证两个替换操作

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "Hello World ." ;  System.out.println(str.replaceAll("l","\_")) ; // He\_\_o Wor\_d .  System.out.println(str.replaceFirst("l","\_")) ; // He\_lo World .  }  } |

**3.2.6、字符串截取**

Oracle的截取是substr()函数，这个函数在使用的时候下标从0或1开始都是可以的，不过程序之中只能从0开始，而且不能设置为负数。

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **方法名称** | **类型** | **描述** |
| 1 | public String substring(int beginIndex) | 普通 | 从指定位置截取到结尾 |
| 2 | public String substring(int beginIndex, int endIndex) | 普通 | 截取指定范围的内容 |

**范例：**字符串截取

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "Hello World ." ;  System.out.println(str.substring(6)) ;  System.out.println(str.substring(0,5)) ;  }  } |

**3.2.7、字符串拆分**

所谓的拆分操作指的就是按照一个指定的字符串标记，对一个完整的字符串进行分割。如果要完成拆分操作，可以使用的方法如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **方法名称** | **类型** | **描述** |
| 1 | public String[] split(String regex) | 普通 | 按照指定的字符串全拆分 |
| 2 | public String[] split(String regex, int limit) | 普通 | 拆分为指定的长度 |

**范例：**完成全拆分

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "Hello World !!!" ;  String result [] = str.split(" ") ;  for (int x = 0 ; x < result.length ; x ++) {  System.out.println(result[x]) ;  }  }  } |

**范例：**拆分为指定的个数

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "Hello World !!!" ;  String result [] = str.split(" ",2) ;  for (int x = 0 ; x < result.length ; x ++) {  System.out.println(result[x]) ;  }  }  } |

如果设置了拆分的个数，那么后面的内容将作为整体不再拆分。

**范例：**现在要求拆分IP地址

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "192.168.1.1" ;  String result []= str.split("\\.") ;  for (int x = 0 ; x < result.length ; x ++) {  System.out.println(result[x]) ;  }  }  } |

**提示：**以后在进行字符串拆分的时候，如果遇见拆不开的问题，就使用“\\”

**3.2.8、其他方法**

以上是一些可以分类的功能性的方法，但是在String类中还有一些无法分类的方法，下面统一列出。

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **方法名称** | **类型** | **描述** |
| 1 | public boolean isEmpty() | 普通 | 判断是否为空字符串（**""**） |
| 2 | public int length() | 普通 | 取得字符串长度 |
| 3 | public **String** trim() | 普通 | 去掉左右空格 |
| 4 | public String toLowerCase() | 普通 | 将全部字符串转小写 |
| 5 | public String toUpperCase() | 普通 | 将全部字符串转大写 |
| 6 | public String intern() | 普通 | 入池 |
| 7 | public String concat(String str) | 普通 | 字符串连接 |

**范例：**取得字符串长度和是否为空

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "hello" ;  System.out.println(str.isEmpty()) ; // false  System.out.println("".isEmpty()) ; // true  System.out.println(str.length()) ;  System.out.println(" Hello ".length()) ; // 空格也计算  }  } |

**范例：**使用trim()去掉空格

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = " hello " ;  System.out.println("字符串内容：〖" + str + "〗") ;  System.out.println("字符串内容：〖" + str.trim() + "〗") ;  // 对象.方法().方法().方法()...代码链，看每个方法的返回值  System.out.println("字符串长度：" + **str.trim().length()**) ;  }  } |
| String s = str.trim () ;  int len = s.length() ;  System.out.println("字符串长度：" + len) ; |

**范例：**其他操作

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "Hello World !~!!" ;  System.out.println(str.toUpperCase());  System.out.println(str.toLowerCase());  System.out.println("Hello ".concat("World .")); // +也可以  }  } |

在Oracle的学习过程之中，发现里面有一个initcap()的方法，可以让首字母大写，可是这个方法其实很重要，但是String类没有提供。下面简单实现一下，给一个基本的原理。

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "hello" ;  System.out.println(initcap(str)) ;  }  public static String initcap(String s) {  return s.substring(0,1).toUpperCase().concat(s.substring(1)) ;  }} |

提示：虽然JDK没有提供此类方法，可是以后学习的Apache的commones组件包之中，有此方法。

**3.2.9、思考题**

1、 现在给出了如下一个字符串格式：“姓名:成绩|姓名:成绩|姓名:成绩”，例如：给定的字符串是：“Tom:90|Jerry:80|Tony: 89”，要求可以对数据进行处理，将数据按照如下的形式显示：

· 例如：显示格式 姓名：Tom，成绩是：90；

给出的数据格式都是有一定的标准的，那么只需要完成拆分即可，拆两次，一次是按照“|”，另外一次是针对每一个“:”完成。

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String str = "Tom:90|Jerry:80|Tony:89" ;  String result [] = str.split("**\\**|") ;  for (int x = 0 ; x < result.length ; x ++) {  String temp [] = result[x].split(":") ;  System.out.println("姓名：" + temp[0] + "，年龄：" + temp[1]) ;  }  }  } |

2、 给定一个email地址，要求验证其是否正确，提示：可以简单的验证一下，重点验证@和.。

既然现在要验证，则给出几个验证的标准：

· 最短的email长度也是5：a@a.a；

· @和.不能作为开头和结尾；

· @和.顺序要有定义，假设只有一个.；

|  |
| --- |
| public class StringDemo {  public static void main(String args[]) {  String email = "aa@aa.com" ;  System.out.println(isEmail(email)) ;  }  public static boolean isEmail(String email) {  if (email == null || email.length()<5) {  return false ;  }  if(email.startsWith("@") || email.startsWith(".")  || email.endsWith("@") || email.endsWith(".")) {  return false ;  }  if (email.indexOf("@") > email.indexOf(".")) {  return false ;  }  return true ;  }  } |

**3.3、this关键字（重点）**

首先需要提醒的是，在整个Java之中，this是最麻烦一个关键字，只要是代码开发，几乎都离不开this。在Java中this可以完成三件事情：表示**本类属性、表示本类方法**、**当前对象（只是先介绍概念）**。

**3.3.1、“this.属性”表示本类属性**

在讲解这一操作之前，首先来观察如下的一段程序。

|  |
| --- |
| class Person {  private String name ;  private int age ;  **public Person(String n,int a) {**  **name = n ;**  **age = a ;**  **}** // setter、getter略  public String getInfo() {  return "姓名：" + name + "，年龄：" + age ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = new Person("张三",20) ;  System.out.println(per.getInfo()) ;  }  } |

但是现在来观察一下构造方法：

|  |
| --- |
| public Person(String n,int a) {  name = n ;  age = a ;  } |

这个时候的构造方法的两个参数的目的是为类中的name和age两个属性初始化，可是这个方法上的两个参数，一个是字母n，另外一个是字母a，什么意思？那么最好的做法，既然构造方法的两个参数是为了name和age属性初始化使用的，最好将其的名称也定义为name和age才最为合适。

|  |
| --- |
| public Person(String name,int age) {  name = name ;  age = age ;  } |

此时构造方法上的两个参数的名称变得有意义了。但是这样一来，一个问题就出现了，发现属性没有内容了。因为在程序之中是以“{}”作为一个分界，采用就近的取用原则，所以现在为了可以明确的指定要操作的是类中属性的话，那么应该采用“this.属性”的形式完成，代码应该变为：

|  |
| --- |
| class Person {  private String name ;  private int age ;  public Person(String name,int age) {  this.name = name ;  this.age = age ;  } // setter、getter略  public String getInfo() {  return "姓名：" + this.name + "，年龄：" + this.age ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = new Person("张三",20) ;  System.out.println(per.getInfo()) ;  }  } |

**提示：在日后的所有开发之中，只要是调用类中属性的情况，都要使用“this.属性”的方式来进行表示。**

**3.3.2、调用本类方法**

对于一个类之中的方法现在是分为两种：

· 普通方法：在之前强调过，如果现在要调用的是本类方法，则可以使用“this.方法()”调用；

· 构造方法：调用其他构造使用“this()”调用。

例如：现在一个类之中存在了三个构造方法（无参、有一个参数、有两个参数），但是不管使用何种构造方法，都要求在实例化对象产生的时候输出一行提示信息：“一个新的类对象被实例化”（假设这个信息等于50行代码），按照之前的学习，代码编写如下：

|  |
| --- |
| class Person {  private String name ;  private int age ;  public Person() {  **System.out.println("\*\*\* 一个新的Person类对象被实例化。") ;**  }  public Person(String name) {  **System.out.println("\*\*\* 一个新的Person类对象被实例化。") ;**  **this.name = name ;**  }  public Person(String name,int age) {  **System.out.println("\*\*\* 一个新的Person类对象被实例化。") ;**  **this.name = name ;**  this.age = age ;  } // setter、getter略  public String getInfo() {  return "姓名：" + this.name + "，年龄：" + this.age ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = new Person("张三",20) ;  System.out.println(per.getInfo()) ;  }  } |

不过，遗憾的是，按照之前所学习到的知识来讲，此时的程序之中会出现大量的重复代码，而我们的目标是没有重复，会偷懒是种美德。这种情况下就可以利用this()来完成。

|  |
| --- |
| class Person {  private String name ;  private int age ;  public Person() {  System.out.println("\*\*\* 一个新的Person类对象被实例化。") ;  }  public Person(String name) {  **this() ; // 调用无参构造**  this.name = name ;  }  public Person(String name,int age) {  **this(name) ; // 调用有一个参数的构造**  this.age = age ;  } // setter、getter略  public String getInfo() {  return "姓名：" + this.name + "，年龄：" + this.age ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = new Person("张三",20) ;  System.out.println(per.getInfo()) ;  }  } |

使用this()就完成了构造方法之间的互相调用操作。

**注意：**在使用this调用构造方法的时候有以下一些问题：

· 所以的构造方法是在对象实例化的时候被默认调用，而且是在调用普通方法之前调用，所以使用“this()”调用构造方法的操作，一定要放在构造方法的首行；

· 如果一个类之中存在了多个构造方法的话，并且这些构造方法都使用了this()互相调用，那么至少要保留一个构造方法没有调用其他构造，以作为程序的出口。

**范例：**观察如下程序

|  |
| --- |
| class Person {  private String name ;  private int age ;  public Person() {  **this("",10) ; // 调用两个参数的构造**  System.out.println("\*\*\* 一个新的Person类对象被实例化。") ;  }  public Person(String name) {  **this() ; // 调用一个参数的构造**  this.name = name ;  }  public Person(String name,int age) {  **this(name) ; // 调用有一个参数的构造**  this.age = age ;  } // setter、getter略  public String getInfo() {  return "姓名：" + this.name + "，年龄：" + this.age ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = new Person("张三",20) ;  System.out.println(per.getInfo()) ;  }  } |

**思考题：**观察构造方法互相调用的操作

写一个公司员工类

数据成员：

员工号、姓名、薪水、部门

方法：

1、 利用构造方法完成设置信息：

A、单参，只传递员工号，则员工姓名：无名氏，薪水：0，部门：未定

B、双参，传递员工号，姓名，则员工薪水为1000，部门：后勤

C、四参，传递员工号，姓名，部门，薪水

D、无参，则均为空值

2、 显示信息

|  |
| --- |
| class Emp {  private int empno ;  private String ename ;  private double salary ;  private String dept ;  public Emp(){}  public Emp(int empno){  **this.empno = empno ;**  **this.ename = "无名氏" ;**  **this.salary = 0.0 ;**  **this.dept = "未定" ;**  }  public Emp(int empno,String ename){  **this.empno = empno ;**  **this.ename = ename ;**  **this.salary = 1000.0 ;**  **this.dept = "后勤" ;**  }  public Emp(int empno,String ename,double salary,String dept){  **this.empno = empno ;**  **this.ename = ename ;**  **this.salary = salary ;**  **this.dept = dept ;**  }  public String getInfo() {  return "雇员编号：" + this.empno + "，姓名：" + this.ename + "，工资：" + this.salary + "，部门：" + this.dept ;  }  } |

如果按照以上的方式开发，是可以完成功能的实现，但是代码之中却会存在重复的代码，很明显，这样不合适。

|  |
| --- |
| class Emp {  private int empno ;  private String ename ;  private double salary ;  private String dept ;  public Emp(){}  public Emp(int empno){  **this(empno,"无名氏",0.0,"未定") ;**  }  public Emp(int empno,String ename){  **this(empno,ename,1000.0,"后勤") ;**  }  public Emp(int empno,String ename,double salary,String dept){  this.empno = empno ;  this.ename = ename ;  this.salary = salary ;  this.dept = dept ;  }  public String getInfo() {  return "雇员编号：" + this.empno + "，姓名：" + this.ename + "，工资：" + this.salary + "，部门：" + this.dept ;  }  }  public class TestDemo {  public static void main(String args[]) {  Emp emp = new Emp(7369,"SMITH") ;  System.out.println(emp.getInfo()) ;  }} |

总结：这种构造方法的互相调用是在对象实例化的时候，不同的构造有一些相同操作的情况下去使用。

**3.3.3、this表示当前对象（暂时了解）**

当前对象指的是当前正在调用本类方法的操作对象，下面首先来验证一下。

|  |
| --- |
| class Demo {  public void print() {  System.out.println("当前对象：" + **this**) ;  }  }  public class TestDemo {  public static void main(String args[]) {  Demo demo1 = new Demo() ;  Demo demo2 = new Demo() ;  System.out.println(demo1) ;  demo1.print() ;  System.out.println("====================") ;  System.out.println(demo2) ;  demo2.print() ;  }  } |

那么如果清楚了当前对象的概念，之前的“this.属性”就表示当前对象的属性。

**3.4、复习引用传递（重点）**

对于引用传递，是整个Java之中的精髓所在，所以清楚这之中的各个操作关系，是最为重要的，下面通过三个程序进行依次的分析。

**范例：**程序一

|  |
| --- |
| class Demo {  private int data = 10 ;  public Demo(int data) {  this.data = data ;  }  public void setData(int data) {  this.data = data ;  }  public int getData() {  return this.data ;  }  }  public class TestDemo {  public static void main(String args[]) {  Demo demo = new Demo(100) ;  fun(demo) ; // Demo temp = demo ;  System.out.println(demo.getData()) ;  }  public static void fun(Demo temp) {  temp.setData(30) ;  }  } |

在使用内存关系图表示的手，首先来换个角度看程序，实际上以上的方法传递过程，等价于以下程序代码：

|  |
| --- |
| public static void main(String args[]) {  Demo demo = new Demo(100) ;  **Demo temp = demo ;**  **temp.setData(30) ;**  System.out.println(demo.getData()) ;  } |

**再次提醒：堆内存保存的是类中的属性，而栈保存的是堆内存的地址（简单理解为对象名称）。**

**范例：**第二道引用传递

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  String str = "Hello" ;  fun(str) ; // String temp = str ;  System.out.println(str) ;  }  public static void fun(String temp) {  temp = "World" ;  }  } |

**本程序解决的关键：String的内容一旦声明则不可改变，改变的是内存地址的指向。**

如果觉得以上的字符串的内存分析方法实在是过于麻烦了，那么换种方式理解，基本数据类型的数据传递不牵扯到内存关系。可以按照如下的方式理解。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data = 10 ;  fun(data) ;  System.out.println(data) ;  }  public static void fun(int temp) {  temp = 20 ;  }  } |

**范例：**第三道引用传递范例

|  |
| --- |
| class Demo {  private String data ;  public Demo(String data) {  this.data = data ;  }  public void setData(String data) {  this.data = data ;  }  public String getData() {  return this.data ;  }  }  public class TestDemo {  public static void main(String args[]) {  Demo demo = new Demo("Hello") ;  fun(demo) ; // Demo temp = demo ;  System.out.println(demo.getData()) ;  }  public static void fun(Demo temp) {  temp.setData("World") ;  }  } |

简单来讲，以上的程序和第一道程序没有区别（简单理解）。

以上的程序与第一道程序最大的不同在于：第一道程序使用的data是int型数据，int数据基本数据类型，但是本题目使用的是String型数据（引用数据类型），不过考虑到String类型的特殊性，可以将字符串想象成基本数据类型那样来理解并且操作，而实际上更完整的画法：

Demo对象（栈）之中包含了String的引用（data是String的名字在栈，而字符串的内容是在堆）不过在Demo对象的堆内存里面保存了一个data（栈内存）的引用关系。

## 3.5、对象比较（重点）

如果说现在有以下的一个程序：

|  |
| --- |
| class Person {  private String name ;  private int age ;  public Person(String name,int age) {  this.name = name ;  this.age = age ;  }  public void setName(String name) {  this.name = name ;  }  public void setAge(int age) {  this.age = age ;  }  public String getName() {  return this.name ;  }  public int getAge() {  return this.age ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per1 = new Person("张三",20) ;  Person per2 = new Person("张三",20) ;  }  } |

请问，如果现在要想比较两个对象是否相等（per1和per2）该怎么做？

现在应该将对象之中的每一个属性进行判断，如果全部属性都相同了，那么表示相等。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  Person per1 = new Person("张三",20) ;  Person per2 = new Person("张三",20) ;  **if (per1.getName().equals(per2.getName())**  **&& per1.getAge() == per2.getAge()) {**  **System.out.println("是同一个对象。") ;**  **} else {**  **System.out.println("不是同一个对象。") ;**  **}**  }  } |

现在的功能已经实现了，但是这样实现有什么问题呢？

|  |  |
| --- | --- |
|  | **通缉令** |
| 姓名：张涛  年龄：50  籍贯：未知  犯的事：………..  奖金：$9999999  三日内发放。 |

以上的代码在进行对象比较的过程之中，采用了客户端（第三方，主方法或者说主类可以理解为客户端）完成的判断，很明显不合适。这种比较的操作应该是每一个对象自己所应该具备的功能。

|  |
| --- |
| class Person {  private String name ;  private int age ;  public Person(String name,int age) {  this.name = name ;  this.age = age ;  }  // 一个类可以接收本类引用  // 接收完成之后可以直接调用本类中的私有操作  public boolean compare(Person per) {  if (per == null) {  return false ; // 直接返回false  }  if (this == per) { // 地址相同  return true ;  }  // 这样一来在compare()方法之中有两个对象：传入的Person，另外一个是this  if (this.name.equals(per.name)  && this.age == per.age) {  return true ;  }  return false ;  }  public void setName(String name) {  this.name = name ;  }  public void setAge(int age) {  this.age = age ;  }  public String getName() {  return this.name ;  }  public int getAge() {  return this.age ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per1 = new Person("张三",20) ;  Person per2 = new Person("张三",20) ;  if (per1.compare(per1)) {  System.out.println("是同一个对象。") ;  } else {  System.out.println("不是同一个对象。") ;  }  }  } |

对象比较的操作一定是一个类自己本身所具备的功能，而且对象比较的操作特点：

· 本类接收自己的引用，而后与本类当前对象（this）进行比较；

· 首先为了避免NullPointerException的产生，应该增加一个null的判断；

· 为了防止浪费性能的情况出现（要判断的属性会多），可以增加地址数值的判断，因为相同的对象地址相同；

· 之后进行属性的依次比较，如果属性全部相同，则返回true，否则返回false。

# 4、总结

1、 String类的两种实例化方式及区别、字符串内容的两种比较操作方法；

2、 String类之中定义的操作方法记住；

3、 this关键字的作用：本类属性、本类方法、当前对象；

4、 对象比较操作（核心）；

5、 简单的复习了引用传递的操作。

# 5、预习任务

static关键字、内部类。

# 6、作业

1、 为十个简单java类增加对象比较操作方法，写熟练为止，属性不能少。

2、 背String类的方法。

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **方法名称** | **类型** | **描述** |
| 1 | public String(char[] value) | 构造 | 将全部的字符数组内容变为字符串 |
| 2 | public String(char[] value, int offset, int count) | 构造 | 将部分字符数组变为字符串，offset表示开始点，count表示要操作的长度 |
| 3 | public char charAt(int index) | 普通 | 取得指定索引位置上的字符 |
| 4 | public char[] toCharArray() | 普通 | 将字符串转换为字符数组 |
| 5 | public String(byte[] bytes) | 构造 | 将全部的字节数组变为字符串 |
| 6 | public String(byte[] bytes, int offset, int length) | 构造 | 将部分的字节数组变为字符串 |
| 7 | public byte[] getBytes() | 普通 | 将字符串变为字节数组 |
| 8 | public byte[] getBytes(String charsetName) throws UnsupportedEncodingException | 普通 | 字符串转码操作 |
| 9 | public boolean equals(String anObject) | 普通 | 区分大小写的相等判断 |
| 10 | public boolean equalsIgnoreCase(String anotherString) | 普通 | 不区分大小写比较是否相等 |
| 11 | public int compareTo(String anotherString) | 普通 | 比较两个字符串的大小 |
| 12 | public boolean contains(String s) | 普通 | 查找指定的子字符串是否存在，JDK 1.5之后有 |
| 13 | public int indexOf(String str) | 普通 | 从头查找指定字符串的位置，找不到返回-1 |
| 14 | public int indexOf(String str, int fromIndex) | 普通 | 由指定位置向后查找字符串的位置，找不到返回-1 |
| 15 | public int lastIndexOf(String str) | 普通 | 由后向前查找字符串的位置，找不到返回-1 |
| 16 | public int lastIndexOf(String str, int fromIndex) | 普通 | 从指定位置由后向前查找 |
| 17 | public boolean startsWith(String prefix) | 普通 | 判断是否以指定的字符串开头 |
| 18 | public boolean startsWith(String prefix, int toffset) | 普通 | 从指定位置判断是否以指定字符串开头，JDK 1.7 |
| 19 | public boolean endsWith(String suffix) | 普通 | 判断是否以指定的字符串结尾 |
| 20 | public String replaceAll(String regex, String replacement) | 普通 | 全部替换 |
| 21 | public String replaceFirst(String regex, String replacement) | 普通 | 替换首个 |
| 22 | public String substring(int beginIndex) | 普通 | 从指定位置截取到结尾 |
| 23 | public String substring(int beginIndex, int endIndex) | 普通 | 截取指定范围的内容 |
| 24 | public String[] split(String regex) | 普通 | 按照指定的字符串全拆分 |
| 25 | public String[] split(String regex, int limit) | 普通 | 拆分为指定的长度 |
| 26 | public boolean isEmpty() | 普通 | 判断是否为空字符串（**""**） |
| 27 | public int length() | 普通 | 取得字符串长度 |
| 28 | public **String** trim() | 普通 | 去掉左右空格 |
| 29 | public String toLowerCase() | 普通 | 将全部字符串转小写 |
| 30 | public String toUpperCase() | 普通 | 将全部字符串转大写 |
| 31 | public String intern() | 普通 | 入池 |
| 32 | public String concat(String str) | 普通 | 字符串连接 |

# 7、测试题讲解

1、 写出Java的数据类型划分及默认值

· 基本数据类型：

|- 数值型：

|- 整型：byte、short、int、long； 🡺 0

|- 浮点型：float、double； 🡺 0.0

|- 布尔型：boolean； 🡺 false

|- 字符型：char； 🡺 '\u0000'

· 引用数据类型：数组、类、接口； 🡺 null

2、 请解释&和&&、|和||的区别

· 逻辑运算：

|- &表示所有的条件都要进行验证，&&属于短路与，如果之前的条件返回了false，后面不再判断；

|- |表示所有的条件都要验证，||属于短路或，如果之前的条件返回了true，后面不再判断；

· 位运算：&表示位与、|表示位或。

3、 请编写一个数组排序的操作；

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {1,6,2,7,8} ;  for (int x = 0 ; x < data.length ; x ++) {  for (int y = 0 ; y < data.length - 1 ; y ++) {  if (data[y] > data[y + 1]) {  int temp = data[y] ;  data[y] = data[y + 1] ;  data[y + 1] = temp ;  }  }  }  }  } |

也可以使用java.util.Arrays.sort()排序。

4、 [SQL]查询每个雇员的编号、姓名、职位、基本工资、部门名称、工资等级；

|  |
| --- |
| SELECT e.empno,e.ename,e.job,e.sal,d.dname,s.grade  FROM emp e,dept d,salgrade s  WHERE e.deptno=d.deptno AND e.sal BETWEEN s.losal AND s.hisal ; |

5、 [SQL]统计出每个部门的编号、名称、位置、雇员人数、平均工资、平均服务年限。

|  |
| --- |
| SELECT d.deptno,d.dname,d.loc,temp.count,temp.avg,temp.avgyear  FROM dept d, (  SELECT deptno dno,COUNT(empno) count,AVG(sal) avg,  AVG(MONTHS\_BETWEEN(SYSDATE,hiredate)/12) avgyear  FROM emp  GROUP BY deptno) temp  WHERE d.deptno=temp.dno(+) ; |

错的每一个答案抄五遍。

# 8、问题回答

1、 二维数组不会

二维数组是属于一张数据表的形式，如果是数据表，应该表示的行和列两个概念，通过行和列才可以定位出一个数据，观察下面的图象：

如果现在要想确认3的索引，则行的索引是0，列的索引2，行和列表示出数据。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [][] = new int [][] {  {1,2,3},{2,3,4},{3,4,5}} ;  for (int x = 0 ; x < data.length ; x ++) {  for (int y = 0 ; y < data[x].length ; y ++) {  System.out.print(data[x][y] + "、") ;  }  System.out.println() ;  }  }  } |

日后的开发绝对99%不会出现二维数组。

2 public class PrintNum{

public static void main(String args[]){

print(60);

}

public static double getNum(double num){

if(num == 1){

return 1;

}

double result = 0;

for(double i= 1;i <= num;i++){

result = num \* getNum(num-1);

}

return result;

}

public static void print(double num){

int sum = 0;

for(double i = 1;i <= num;i++){

sum+=getNum(i);

}

System.out.println(sum);

}

}

//上面的方法用3,4测试可以但改成60就卡住了不出结果，请问老师什么原因？

3、 public class TestDemo {

public static void main(String args[]) {

int data [] = new int [] {209,201,2,2,3,6,7} ;

int searchData = 3 ; // 要查找的内容

boolean flag = false ; // 查找标记

// flag = true：表示查找到内容

// flag = false：表示没有查找到内容

for (int x = 0 ; x < data.length ; x ++) {

if (searchData == data[x]) { // 找到了

flag = true ; // 找到

break ; // 退出循环

}

}

if (flag) { // 为true

System.out.println("已经查找到了内容。") ;

} else {

System.out.println("没有找到内容。") ;

}

}

}