

# Java 程序设计

## 第 8 章 常用实用类





# 导读

## □ 主要内容

- ◆ String 类、 StringTokenizer 类、 StringBuffer 类
- ◆ Scanner 类
- ◆ Date 与 Calendar 类、日期格式化
- ◆ Math 、 BigInteger 与 Random 类、数字格式化
- ◆ Class 类与 Console 类
- ◆ Pattern 与 Match 类

## □ 重点和难点

- ◆ 重点：字符串的常用方法；字符串分析器使用；类 Date 和类 Calendar 以及类 Math 的使用
- ◆ 难点：字符串分析器的使用；各常用类的实际运用





# 8.1 String 类

- Java 专门提供了用来处理字符序列的 **String 类**
- **String 类在 java.lang 包中**，由于 java.lang 包中的类被默认引入，因此**程序可以直接使用 String 类**
- 需要注意的是 Java 把 String 类声明为 final 类，因此**用户不能扩展 String 类**，即 String 类不可以有子类。



## 8.1.1 构造字符串对象

1. 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。

Java 把用户程序中的 String 常量放入常量池。因为 String 常量是对象，所以也有自己的引用和实体，如图 8.1 所示。例如 String 常量对象 "你好" 的引用是 12AB，实体里是字符序列：你好。

可以这样简单的理解常量池：常量池中的数据在程序运行期间再也不允许改变

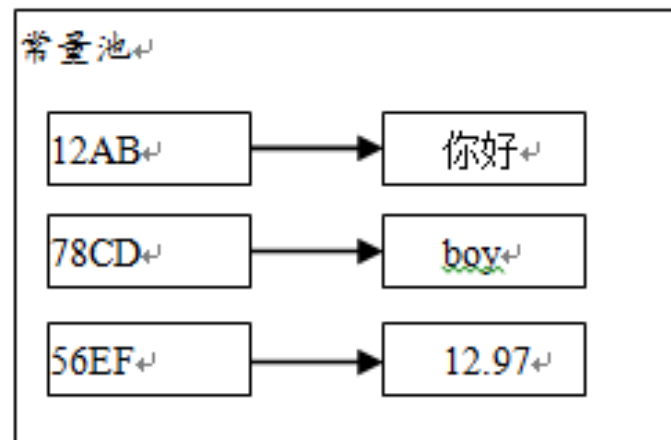


图 8.1 常量池中的常量



# 8.1.1 构造字符串对象

## 2. String 对象

可以使用 String 类声明对象并创建对象，例如：

```
String s = new String("we are students");
```

```
String t = new String("we are students");
```

$s == t$  ???

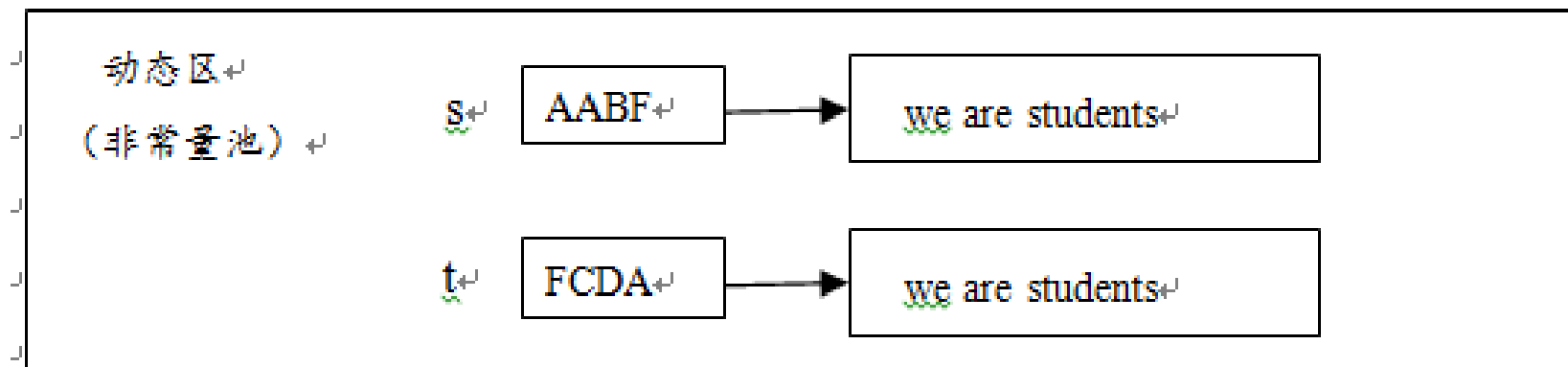


图 8.2 创建字符串对象





## 8.1.1 构造字符串对象

另外，用户无法输出 String 对象的引用：

```
System.out.println(s);
```

输出的是对象的实体，即字符序列 `we are students` 。

也可以用一个已创建的 String 对象创建另一个 String 对象，  
如：

```
String tom = new String(s);
```



## 8.1.1 构造字符串对象

String 类还有两个较常用的构造方法。

(1) **String(char a[])** 用一个字符数组 a 创建一个 String 对象, 如:

```
char a[] = {'J','a','v','a'};
```

```
String s = new String(a);
```

过程相当于

```
String s = new String("Java");
```

(2) **String(char a[],int startIndex,int count)** 提取字符数组 a 中的一部分字符创建一个 String 对象, 参数 startIndex 和 count 分别指定在 a 中提取字符的起始位置和从该位置开始截取的字符个数:

```
char a[] =
```

```
{ '零','壹','贰','叁','肆','伍','陆','柒','捌','玖' };
```

```
String s = new String(a,2,4);
```

相当于



# 8.1.1 构造字符串对象

## 3. 引用 String 常量

String 常量是对象，因此可以把 String 常量的引用赋值给一个 String 对象，例如

```
String s1,s2;  
s1 = " 你好 ";  
s2 = " 你好 ";
```

s1,s2 具有相同的引用（12AB），表达式 s1==s2 的值是 true，因而具有相同的实体。s1，s2 内存示意如图 8.3 所示。

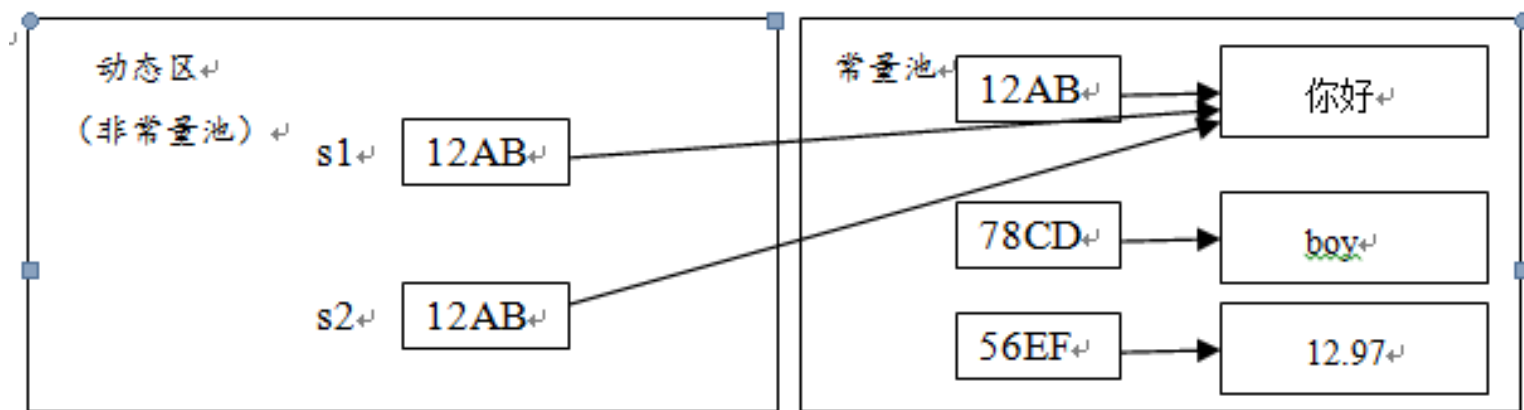


图 8.3 String 常量赋值给 String 对象





## 8.1.1 构造字符串对象

由于用户程序无法知道常量池中“你好”的引用，那么把 String 常量的引用赋值给一个 String 对象 s1 时，Java 让用户直接写常量的实体内容来完成这一任务，但实际上赋值到 String 对象 s1 中的是 String 常量“你好”的引用（见图 8.3）。s1 是用户声明的 String 对象，s1 中的值是可以被改变的，如果再进行 s1 = “boy”，那么 s1 中的值将发生变化（变成 78CD）。

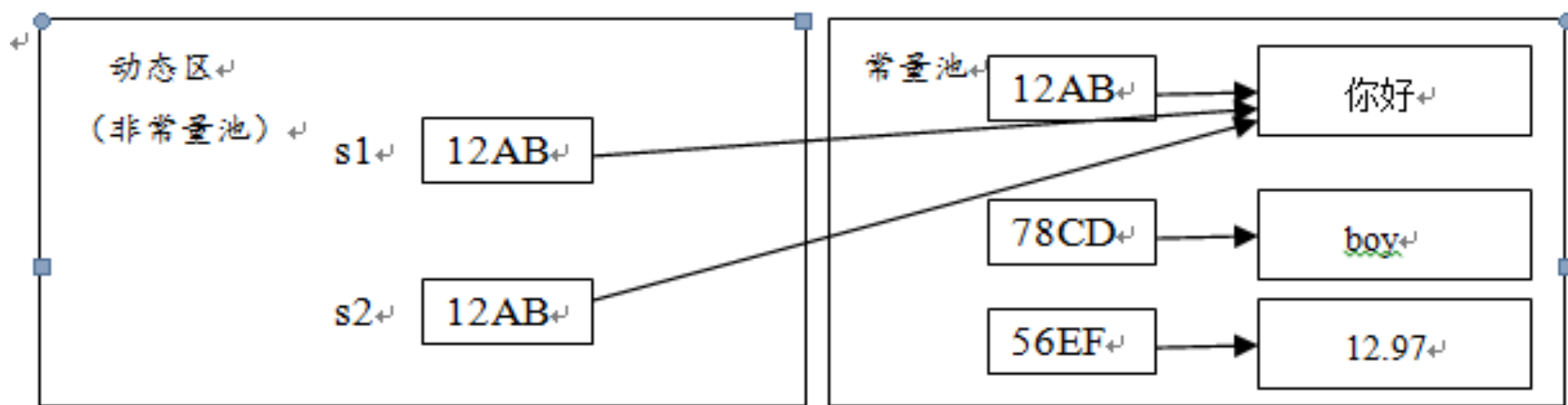


图 8.3 String 常量赋值给 String 对象



## 8.1.2 字符串的并置

String 对象可以用 “+” 进行并置运算，即首尾相接得到一个新的 String 对象。

```
String you = "你";  
String hi = "好";  
String testOne;
```

you 和 hi 进行并置运算 you+hi 得到一个新的 String 对象，可以将这个新的 String 对象的引用赋值给一个 String 声明的变量 testOne = you+hi;

如果是两个常量进行并置运算，那么得到的仍然是常量，如果常量池没有这个常量就放入常量池。

“你”+“好”  
的结果就是常量池中的  
“你好”

那么 testOne 的实体中的字符序列是 “你好”。需要注意的是，参与并置运算的 String 对象，只要有一个是变量，那么 Java 就会在动态区存放所得到的新 String 对象的实体和引用，you+hi 相当于 new String(“你好”)。



**仔细阅读例子 1，理解程序的输出结果。**

```
public class Example8_1 {  
    public static void main(String args[]) {  
        String hello = " 你好 ";  
        String testOne = " 你 "+" 好 ";          // 【代码 1】  
        System.out.println(hello == testOne); // 输出结果是 true / false  
        System.out.println(" 你好 " == testOne); // 输出结果是 true / false  
        System.out.println(" 你好 " == hello); // 输出结果是 true / false  
        String you = " 你 ";  
        String hi = " 好 ";  
        String testTwo = you+hi;                  // 【代码 2】  
        System.out.println(hello == testTwo); // 输出结果是 true / false  
        String testThree = you+hi;  
        System.out.println(testTwo == testThree); // 输出结果是 true / false  
    }  
}
```



## 例子 1 代码讲解

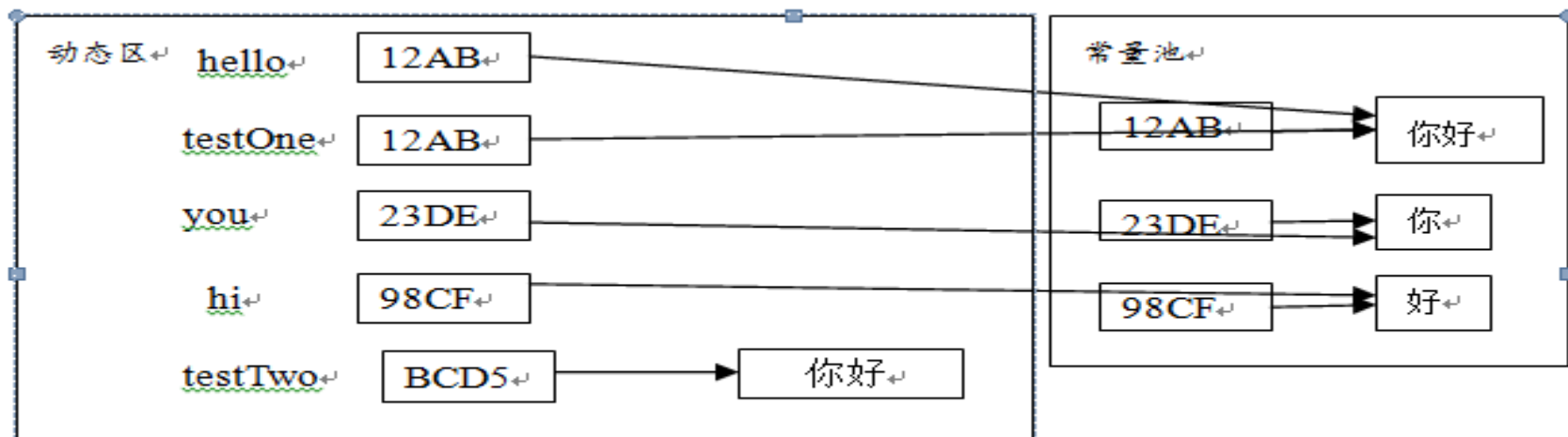
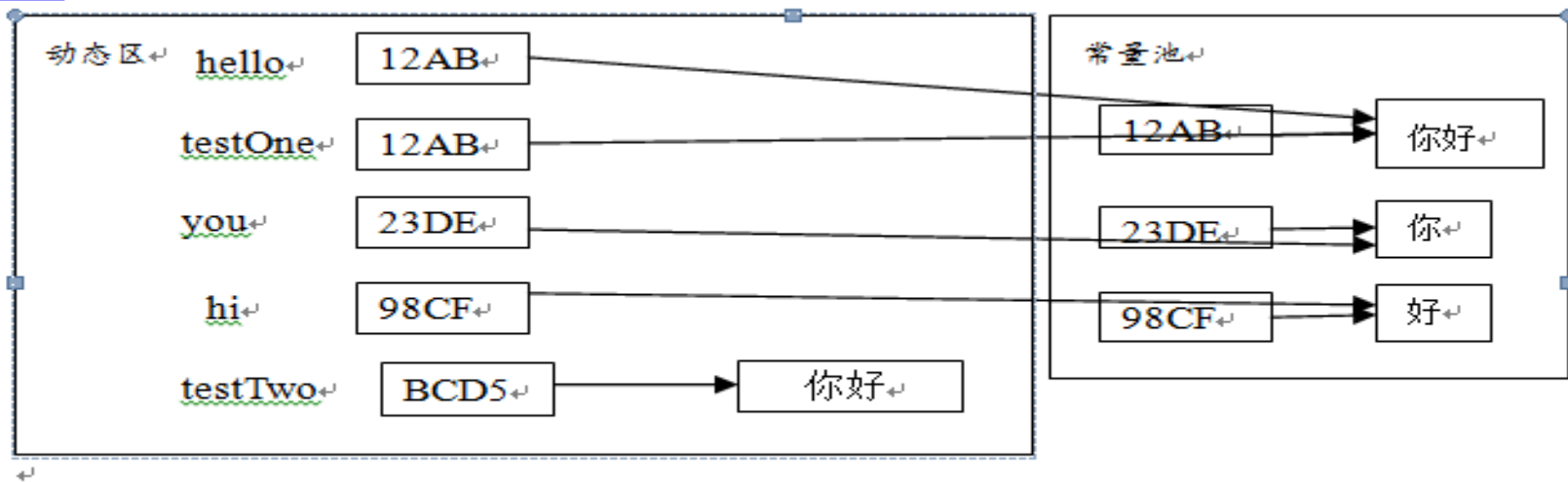


图 8.4 代码讲解示意图

【代码 1】：**String testOne = "你"+"好"**；的赋值号的右边是两个常量进行并置运算，因此，结果是常量池中的常量"你好"（如果读者学习过编译原理，可能知道所谓的常量优化技术，常量折叠是一种 Java 编译器使用的优化技术，**String testOne = "你"+"好"**，被编译器优化为 **String testOne = "你好"**，就像 **int x = 1+2** 被优化为 **int x = 3** 一样），所以表达式 **"你好" == testOne** 和表达式 **hello == testOne** 的值都是 true。



## 例子 1 代码讲解续



【代码 2】：**testTwo = you+hi;** 的赋值号的右边有变量，例如变量 `you` 参与了并置运算，那么 `you+hi` 相当于 `new String("你好")`；因此结果在动态区诞生新对象，`testTwo` 存放着引用 `BCD5`，`testTwo` 的实体中存放者字符序列：你好（如图 8.4），所以表达式 **hello == testTwo** 的结果是 `false`。

## 8.1.2 String 类的常用方法 \_1

- 1 . **public int length():** 获取一个字符串的长度
- 2 . **public boolean equals(String s):** 判断当前 String 对象的字符序列是否与参数 s 指定的 String 对象的字符序列相同

**例题2** 说明了 equals 的用法。

- 3 . **public boolean startsWith(String s)**

判断当前 String 对象的字符序列前缀是否是参数指定的 String 对象 s 的字符序列，例如：

String tom = " 天气预报，阴有小雨 ", jerry = " 比赛结果，中国队赢得胜利 ";

tom.startsWith(“ 天气” ) 的值是 true ；



## 8.1.2 String 类的常用方法 \_2

4. **public int compareTo(String s):** 按字典序与参数 s 指定的**字符序列**比较大小。

如果当前 String 对象的字符序列与 s 的相同，该方法返回值 0，如果大于 s 的字符序列，该方法返回正值；如果小于 s 的字符序列，该方法返回负值。

例如，字符 a 在 Unicode 表中的排序位置是 97，字符 b 是 98，那么对于 **String str = “abcde”; str.compareTo(“boy”)** 小于 0，**str.compareTo(“aba”)** 大于 0，**str.compareTo(“abcde”)** 等于 0

**其相关方法** **public int compareToIgnoreCase(String s)**





**例 题 3** SortString.java Example8\_2.java 使用  
java.util 包中的 **Arrays.sort** 方法和自己编写  
SortString 类中的 sort 方法将一个 String 数组中的  
String 对象的字符序列按字典序排序。

```
public static void sort(String a[]) {  
    for (int i=0; i<a.length-1; i++) {  
        for (int j=i+1; j<a.length; j++) {  
            if (a[j].compareTo(a[i]) < 0) {  
                String temp = a[i];  
                a[i] = a[j];  
                a[j] = temp; } } } }
```





## 8.1.2 String 类的常用方法 \_3

5. **public boolean contains(String s):** String 对象调用 contains 方法判断当前 **String 对象的字符序列** 是否包含参数 **s 的字符序列**，例如， tom="student"，那么 tom.contains("stu") 的值就是 true，而 tom.contains("ok") 的值是 false。



## 8.1.2 String 类的常用方法 \_3

6. **public int indexOf (String str) :** String 对象调用方法从当前 String 对象的字符序列的 0 索引位置开始检索首次出现 str 的字符序列的位置，并返回该位置。如果没有检索到，该方法返回的值是 - 1，**相关方法：**

**indexOf(String s ,int startpoint)**

**lastIndexOf (String s)**

**例如** String tom = "I am a good cat" ;

tom.indexOf("a");// 值是 2

tom.indexOf("good",2);// 值是 7

tom.indexOf("a",7);// 值是 13

tom.indexOf("w",2);// 值是 -1



## 8.1.2 String 类的常用方法 \_3

7. **public String substring(int startpoint):** 字符串对象调用该方法获得一个新的 String 对象，新的 String 对象的字符序列是复制当前 String 对象的字符序列中的 startpoint 位置至最后位置上的字符所得到的字符序列。String 对象调用 **substring(int start, int end)** 方法获得一个新的 String 对象，新的 String 对象的字符序列是复制当前 String 对象的字符序列中的 **start 位置至 end-1 位置**上的字符所得到的字符序列。



## 8.1.2 String 类的常用方法 \_3

- 8 . **public String trim()** : 得到一个新的 String 对象, 这个新的 String 对象的字符序列是当前 String 对象的字符序列去掉前后空格后的字符序列。



## 8.1.3

# 字符串与基本数据的相互转化 \_1

➤ Java.lang 包中的 Integer 类调用其类方法：

**public static int parseInt(String s)**

可以将由“数字”字符组成的字符串，如“876”，转化为 int 型数据，例如：

**int x;**

**String s = “876”;**

**x =**

**Integer.parseInt(s);**

**public static byte parseByte(String s) throws  
NumberFormatException**

**public static short parseShort(String s) throws  
NumberFormatException**

**public static long parseLong(String s) throws**



## 8.1.3

# 字符串与基本数据的相互转化

## 2

➤ 可以使用 `String` 类的类方法

`public static String valueOf(byte n)`

`public static String valueOf(int n)`

`public static String valueOf(long n)`

`public static String valueOf(float`

`n)`  
`public static String valueOf(double n)`

将形如 123 、 1232.98 等数值转化为字符串，

如：

`String str = String.valueOf(12313.9876);`

```
C:\z>java Example8_4 78.86 12 25 125 98  
sum=338.86
```

图 8.4 使用 `main` 方法的参数



**例题4** 求若干个数之和，程序输出结果如图 8.4 。

**注** 应用程序中的 main 方法中的参数 args 能接受用户从键盘键入的字符串。

比如，使用解释器 java.exe 来执行主类

```
C:\ch8\>java Example8_4 78.86 12 25 125 98
```

这时程序中的 args[0] 、 arg[1] 、 arg[2] 、 arg[3] 和 args[4]

分 别 得 到 字 符  
串 “ 78.86” 、 “ 12” 、 “ 25” 、 “ 125” 和 “ 98” 。





## 8.1.4 对象的字符串表示

- 在子类中我们讲过，所有的类都默认是 `java.lang` 包中 `Object` 类的子类或间接子类。 `Object` 类有一个

**`public String toString()`**

方法，一个对象通过调用该方法可以获得该对象的字符序列表示。一个对象调用 `toString()` 方法返回的 `String` 对象的字符序列的一般形式为：

**创建对象的类的名字** @ **对象的引用的字符串表示**

例题 5 [TV.java](#) [Example8\\_5.java](#)

- `TV` 类重写了 `toString()` 方法，并使用 `super` 调用隐藏的 `toString()` 方法







## 8.1.4 对象的字符串表示

```
class TV {  
    double price;  
    public void setPrice(double d) {  
        this.price = d;  
    }  
    public String toString() {  
        String oldStr = super.toString();  
        return oldStr + "\n 这是电视机， 价格是： " +  
price;  
    }  
}
```



## §8.1.5 字符串与字符、字节数组 \_1

### 1. 字符串与字符数组

- String 类的构造方法：
  - `String(char[])` ;
  - `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
- 将 String 对象的部分字符序列存放到数组中的方法：
  - `public void getChars(int start,int end,char c[],int offset )`
- 将 String 对象的字符序列存放到数组中的方法
- `public char[] toCharArray()`

例题6 具体地说明了



抗战胜利  
十一长假期间，学校都放假了

去的使用

图 8.6 字符串与字符数组



## 8.1.5 字符串与字符、字节数组 \_1

### 2. 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()** 使用平台默认的字符编码，将当前 String 对象的字符序列存放到字节数组，并返回数组的引用。
- **public byte[] getBytes(String charsetName)** 使用参数指定字符编码，将当前 String 对象的字符序列存放到字节数组，并返回数组的引用。 。



**例题7** 假设机器的默认编码是： GB2312 。字符串：“Java 你好”调用 `getBytes()` 返回一个字节数组 `d`，其长度为 8，注意该字节数组的 `d[0] ~ d[7]` 单元存放字符的编码的情况。

String 常量：“Java 你好”，调用 `getBytes()` 返回一个字节数组 `d`，其长度为 8，该字节数组的 `d[0]`、`d[1]`、`d[2]` 和 `d[3]` 单元分别是字符 J、a、v 和 a 的编码，第 `d[4]` 和 `d[5]` 单元存放的是字符‘你’的编码（GB2312 编码中，一个汉字占 2 个字节），第 `d[6]` 和 `d[7]` 单元存放的是字符‘好’的编码。程序运行效果如图 8.10。



```
数组d的长度是:8
好
Java你
```

```
byte[] bts = "Java 你好".getBytes();
System.out.println(bts.length);
String s = new String(bts, 6, 2);
System.out.println(s);
s = new String(bts, 0, 6);
System.out.println(s);
```

图 8.10 字符串与字节数组





### 3. 字符串的加密算法

使用一个 String 对象 password 的字符序列作为密码对另一个 String 对象 sourceString 的字符序列进行加密，操作过程如下。

将 password 的字符序列存放到一个字符数组中，

```
char [] p = password.toCharArray();
```

设数组 p 的长度为 n，那么就将待加密的 sourceString 的字符序列按顺序以 n 个字符为一组（最后一组中的字符个数可小于 n），对每一组中的字符用数组 a 的对应字符做加法运算。比如，某组中的 n 个字符是  $a_0a_1\dots a_{n-1}$ ，那么按如下方式

$$c_0 = (\text{char})(a_0 + p[0]), \quad c_1 = (\text{char})(a_1 + p[1]) \dots c_{n-1} = (\text{char})(a_{n-1} + p[n-1])$$

得到加密后的字符串。

上述加密算法的解密算法是对密文做减法运算



# 例子8

在下面的例子 8 中

EncryptAndDecrypt.java, Example8\_8.java

用户输入密码来加密“今晚十点进攻”，运行效果如图 8.11 。

```
输入密码加密:今晚十点进攻  
nihao123  
密文:伸睫厖焚遊敬  
输入解密密码  
nihao123  
明文:今晚十点进攻
```

图 8.11 加密字符串



# 例子8

```
class EncryptAndDecrypt {  
    String encrypt(String source, String password) {  
        char[] c = source.toCharArray();  
        char[] p = password.toCharArray();  
        int m = c.length;  
        int n = p.length;  
        for (int i=0; i<m; i++) {  
            c[i] = (char)(c[i] + p[i%n]); }  
        return new String(c); }  
    String decrypt(String source, String password) {  
        char[] c = source.toCharArray();  
        char[] p = password.toCharArray();  
        int m = c.length;  
        int n = p.length;  
        for (int i=0; i<m; i++) {  
            c[i] = (char)(c[i] - p[i%n]); }  
        return new String(c); }  
}
```



# 8.1.6 正则表达式及字符串的替换与分解\_1

## 1. 正则表达式

正则表达式是一个 **String 对象的字符序列**，该字符序列中含有具有特殊意义字符，这些特殊字符称做正则表达式中的元字符。比如，"`\\dcat`" 中的 `\\d` 就是有特殊意义的元字符，代表 0 到 9 中的任何一个，"`0cat`"，"`1cat`"，"`2cat`"，...，"`9cat`" 都是和正则表达式 "`\\dcat`" 匹配的字符序列。

String 对象调用 **`public boolean matches(String regex)`** 方法可以判断当前 String 对象的字符序列是否和参数 `regex` 指定的正则表达式匹配。[参考表8.1](#)，[表8.2](#)。





## 8.1.6 正则表达式及字符串的替换与分解\_1

在正则表达式中可以用**方括号**括起若干个字符来表示一个元字符，该元字符代表方括号中的任何一个字符。例如：

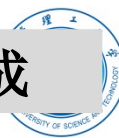
String        regex        =        “[159]ABC”        ,        那么 “1ABC”、 “5ABC” 和 “9ABC” 都是和正则表达式 regex 匹配的字符序列；

[abc] : 代表 a、b、c 中的任何一个；

[^abc] : 代表除了 a、b、c 以外的任何字符；

[a-zA-Z] : 代表英文字母（包括大写和小写）中的任何一个  
任何由英文字母，数字或下划线组成的字符序列都和正则表达式 “**[a-zA-Z|0-9|\_]+**” 匹配

**例9** 程序判断用户从键盘输入的字符序列是由英文字母，数字或下划线所组成



# 8.1.6 正则表达式及字符串的替换与分解 \_1

## 2. 字符串的替换

**public String replaceAll(String regex,String replacement)**

**String 对象调用 public String replaceAll(String regex,String replacement) 方法返回一个新的 String 对象，这个新的 String 对象的字符序列是把当前 String 对象的字符序列中所有和参数 regex 匹配的子字符序列，用参数 replacement 的字符序列替换后得到字符序列**



# 8.1.6 正则表达式及字符串的替换与分解 \_1

## 2. 字符串的替换

```
public String replaceAll(String regex,String replacement)
```

```
String str ="12hello567bird".replaceAll("[a-zA-Z]+"," 你好 ");
```

那么 str 的字符序列就是将 "12hello567bird" 中所有英文字符序列替换为 " 你好 " 后得到的字符序列，即 str 的字符序列是 "12 你好 567 你好"。

例题10 String 对象 **str** 调用 replaceAll() 方法返回一个新的 String 对象，该对象的字符序列是 **str** 的字符序列中的网站链接地址被替换为 “\*\*\*\*\*” 后的字符序列。



# 8.1.6 正则表达式及字符串的替换与分解 \_1

## 2. 字符串的替换

`public String replaceAll(String regex,String replacement)`

```
String str=" 欢迎大家访问 http://www.xiaojiang.cn 了解、参观公  
司 ";  
String regex = "(http://|www)\\56?\\w+\\56{1}\\w+\\56{1}\\p{Alpha}+";  
System.out.printf(" 替换 \\n\\\"%s\\\"\\n 中的网站链接信息后得到的字符  
串: \\n",str);  
str = str.replaceAll(regex, "*****");  
System.out.println(str);
```



# 8.1.6 正则表达式及字符串的替换与分解 \_2

## 3 . 字符串的分解

`public String[] split(String regex)` 使用参数指定的正则表达式 `regex` 做为分隔标记分解出其中的单词，并将分解出的单词存放在字符串数组中。 \_\_\_\_



# 8.1.6 正则表达式及字符串的替换与分解 \_2

## 3. 字符串的分解

```
public String[] split(String regex)
```

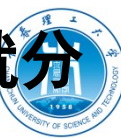
例如，对于字符串 str

```
String str = "1949 年 10 月 1 日是中华人民共和国成立的日子";
```

如果准备分解出全部由数字字符组成的单词，就必须用非数字字符串做为分隔标记，因此，可以使用正则表达式：`String regex="\\D+"`;

做为分隔标记分解出 str 中的单词：`String digitWord[]=str.split(regex);`

那么，`digitWord[0]`、`digitWord[1]` 和 `digitWord[2]` 就分



# 8.1.6 正则表达式及字符串的替换与分解 \_2

## 3. 字符串的分解

`public String[] split(String regex)` 使用参数指定的正则表达式  
regex

例题11 用户从键盘输入一行文本，程序输出其中的单词。

```
一行文本:  
who are you (Caven?)  
单词1:who  
单词2:are  
单词3:you  
单词4:Caven
```

图 8.10 正则表达与字符串的分解





# 注意事项

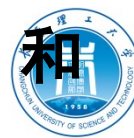
需要**特别注意**的是，`split` 方法认为分隔标记的左面应该是单词，因此如果和当前 `String` 对象的字符序列的前缀和 `regex` 匹配，那么 `split(String regex)` 方法分解出的第一个单词是不含任何字符的字符序列（长度为 0 的字符序列），即 `""`。例如，对于：

```
String str = "公元 1949 年 10 月 1 日是中华人民共和国成立的日子";
```

使用正则表达式 `String regex="\D+"` 作为分隔标记分解 `str` 的字符序列中的单词：

```
String digitWord[]=str.split(regex);
```

那么，数组 `digitWord` 的长度是 4，不是 3。  
`digitWord[0]`、`digitWord[1]`，`digitWord[2]`







## 8. 2 StringTokenizer 类

- 本节学习怎样使用 StringTokenizer 对象分解 String 对象的字符序列。
- **StringTokenizer 类**在 java.util 包中，有两个常用的构造方法：
  - **StringTokenizer(String s)** : 为 String 对象 s 构造一个分析器。使用默认的分隔标记，即空格符、换行符、回车符、Tab 符、进纸符做分隔标记。
  - **StringTokenizer(String s, String delim)** 为 String 对象 s 构造一个分析器。参数 delim 的字符序列中的字符的任意排列被作为分隔标记。





## 8. 2 StringTokenizer 类

例如：

```
StringTokenizer fenxi = new StringTokenizer("you#*are*##welcome", "#*");
```

如果指定字符 # 和字符 \* 是分隔标记，那么字符 # 和字符 \* 的任意排列，比如，####\*#\* 就是一个分隔标记，即 "You#are#\*welcome" 和 "You\*\*\*#are\*##\*#welcome" 都有三个单词，分别是 we，are 和 welcome。



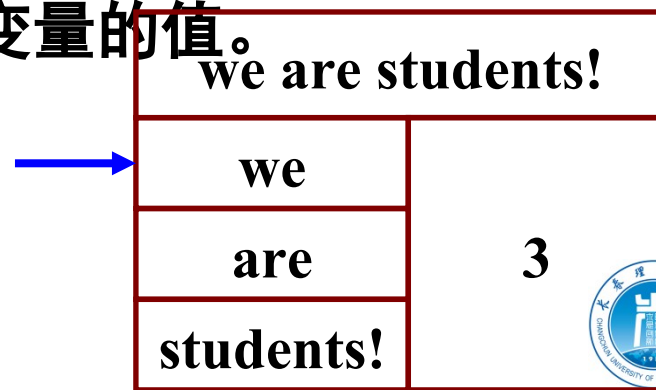
# 具体使用步骤

- **StringTokenizer 对象**称作一个字符串分析器可以使用下列方法：

(1) **nextToken()** : 逐个获取字符串中的语言符号（单词），字符串分析器中的负责计数的变量的值就自动减一。

(2) **hasMoreTokens()** : 只要字符串中还有语言符号，即计数变量的值大于 0，该方法就返回 true，否则返回 false。

(3) **countTokens()** : 得到分析器中计数变量的值。



# 例子 12 计算购物小票的中的商品价格的和

程序关心是购物小票中的数字，因此需要分解出这些数字，以便单独处理，这样就需要把非数字的字符序列替换成统一的字符，以便使用分隔标记分解出数字，例如，对于 "12#25#39.87"，如果用字符 # 做分隔标记，就很容易分解出数字单词。在例子 12 的 PriceToken 类中，把购物小票中非数字的字符序列都替换成 #，然后再分解出数字单词（价格），并计算出这些数字的和，运行效果如图 8.14。

PriceToken.java

Example8\_12.java

```
牛奶:8.5圆, 香蕉3.6圆, 酱油:2.8圆  
购物总价格14.90  
商品数目:3, 平均价格:4.97
```

图 8.14 使用 StringTokenizer 类





## 8.3 Scanner 类

- 使用 Scanner 类从字符串中解析程序所需要的数据。

Scanner 对象可以解析字符序列中的单词，例如，对于 String 对象 NBA

```
String NBA = "I Love This Game";
```

为了解析出 NBA 的字符序列中的单词，可以如下构造一个 Scanner 对象。

```
Scanner scanner = new Scanner(NBA);
```

Scanner 对象可以调用方法

```
useDelimiter( 正则表达式 );
```

将正则表达式作为分隔标记，即让 Scanner 对象在解析操作时，把与正则表达式匹配的字符序列作为分隔标记。如果不指定分隔标记，那么 Scanner 对象默认地用空白字符：空格，制表符，回行符作为分隔标记来解析 String 对象的字符序列中的单词。



## 8.3 Scanner 类

- **scanner 将空格做为分隔标记来解析字符序列中的单词，具体解析操作：**
  - **scanner 调用 next() 方法依次返回 NBA 中的单词，如果 NBA 最后一个单词已被 next() 方法返回， scanner 调用 hasNext() 将返回 false ， 否则返回 true 。**
  - **对于被解析的字符序列中的数字型的单词，比如 618 ， 168.98 等， scanner 可以用 nextInt() 或 nextDouble() 方法来代替 next() 方法，即 scanner 可以调用 nextInt() 或 nextDouble() 方法将数字型单词转化为 int 或 double 数据返回。**
  - **如果单词不是数字型单词， scanner 调用 nextInt() 或 nextDouble() 方法将发生 InputMismatchException 异常，在处理异常时可以调用 next() 方法返回该非数字化单词。**



## 8.3 Scanner 类

### 2. 使用正则表达式作为分隔标记解析字符串

- Scanner 对象可以调用 `useDelimiter( 正则表达式 )`；方法将一个正则表达式作为分隔标记，即和正则表达式匹配的字符串都是分隔标记。

**例题14** 使用正则表达式（匹配所有非数字字符串） `String regex="[^\d.]+"`；作为分隔标记解析字符串："话费清单：市话费 76.89 元，长途话费 167.38 元，短信费 12.68 元" 中的全部价格数字，并计算了总的通信费用。

```
76.89
167.38
12.68
总通信费用:256.95元
```

图 8.12 解析出通信费用



# 例子 13

使用正则表达式：

`String regex= "[^0123456789.]+"` // （匹配所有非数字字符序列）

作为分隔标记，解析 "市话 76.8 元，长途 :167.38 元，短信 12.68 元"，以及 "牛奶 :8.5 圆，香蕉 3.6 圆，酱油 :2.8 圆" 中的价格，并计算价格之和。程序运行效果如图 8.15

Example8\_13.java

GetPrice.java

```
市话76.8元, 长途:167.38元, 短信12.68元  
总价:256.86圆  
牛奶:8.5圆, 香蕉3.6圆, 酱油:2.8圆  
总价:14.90圆
```

第4

8

章

图 8.15 使用 Scanner 类

用类





## 8.4 StringBuffer 类

String 对象的字符序列是不可修改的，也就是说，String 对象的字符序列的字符不能被修改、删除，即 String 对象的实体是不可以再发生变化的，例如，对于

**String s = new String(" 我喜欢散步 ");**

如图 8.16 所示意。

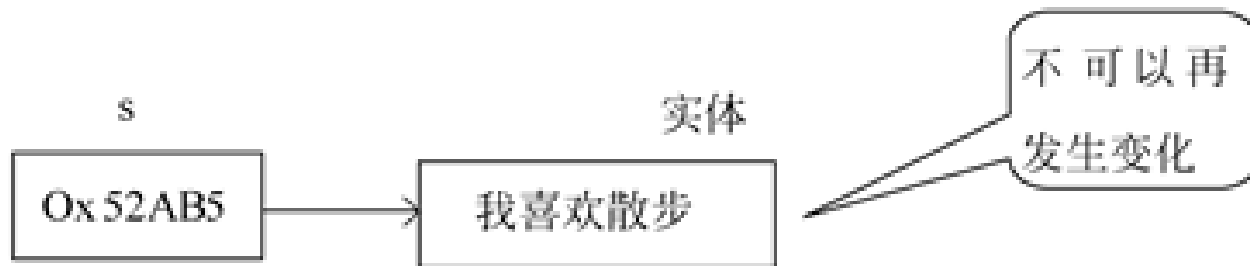


图 8.16 实体不可变



## 8.4 StringBuffer 类

StringBuffer 类的对象的实体的内存空间可以自动地改变大小，便于存放一个可变的字符序列。比如，对于：

```
StringBuffer s = new StringBuffer(" 我喜欢 ");
```

对象 s 可调用 append 方法追加一个字符序列，如图 8.17 。

```
s.append(" 玩篮球 ");
```



图 8.17 实体可变





## 8.4 StringBuffer 类 \_ 续

1) **StringBuffer append(String s):** 将 String 对象 s 的字符序列追加到当前 StringBuffer 对象的字符序列中, 并返回当前 StringBuffer 对象的引用

**StringBuffer append(int n) :** 将 int 型数据 n 转化为 String 对象, 再把该 String 对象的字符序列追加到当前 StringBuffer 对象的字符序列中, 并返回当前 StringBuffer 对象的引用

**StringBuffer append(Object o) :** 将一个 Object 对象 o 的字符序列表示追加到当前 String- Buffer 对象的字符序列中, 并返回当前 StringBuffer 对象的引用

类似的方法还有:

**StringBuffer append(long n), StringBuffer append(boolean n),  
StringBuffer append(float n), StringBuffer append(double n),  
StringBuffer append(char n)**





## 8.4 StringBuffer 类 \_ 续

2)**public char charAt(int n )**: 得到参数 n 指定的置上的单个字符

**public void setCharAt(int n ,char ch)**: 将当前 StringBuffer 对象实体中的字符串位置 n 处的字符用参数 ch 指定的字符替换

3)**StringBuffer insert(int index, String str)** : 将参数 str 指定的字符串插入到参数 index 指定的位置

4)**public StringBuffer reverse()** : 将该对象实体中的字符翻转



## 8.4 StringBuffer 类 \_ 续

**5)StringBuffer delete(int startIndex, int endIndex):** 从当前 StringBuffer 对象实体中的字符串中删除一个子字符串

**其相关方法 :deleteCharAt(int index)** 删除当前 StringBuffer 对象实体的字符串中 index 位置处的一个字符。

**6)StringBuffer replace(int startIndex, int endIndex, String str):**  
将当前 StringBuffer 对象实体中的字符串的一个子字符串用参数 str 指定的字符串替换

**例题14** 使用 StringBuffer 类的常用方法

```
str:大家好  
length:3  
capacity:16  
we好  
we are all好  
we are all right
```

图 8.20 StringBuffer 类的常用方法





# 8.5 Date 与 Calendar 类

## 1. Date 类

- Date 类在 java.util 包中。
- Date 类的构造方法之一：
  - **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。  
例： **Date nowTime=new Date();**
  - 当前 nowTime 对象中含有的日期、时间就是创建 nowTime 对象时的本地计算机的日期和时间。
  - 例如，假设当前时间是：2011 年 3 月 10 日 23:05:32（CST 时区），那么 **System.out.println(nowTime);** 输出结果是：Thu Mar 10 23:05:32 CST 2011。
  - **Date** 对象表示时间的默认顺序是：星期、月、日、小时、分、秒、年。





## 8.5 Date 与 Calendar 类

- Date 类的构造方法之二:

- **Date(long time)** 使用 long 型参数创建指定的时间
- 计算机系统将其自身的时间的“公元”设置在 1970 年 1 月 1 日 0 时（格林威治时间），可以根据这个时间使用 Date 的带参数的构造方法：**Date(long time)** 来创建一个 Date 对象，

例如：**Date date1=new Date(1000), date2=new Date(-1000);**

- 其中的参数取正数表示公元后的时间，取负数表示公元前的时间，其中 1000 表示 1000 毫秒，那么，date1 含有的日期、时间就是计算机系统公元后 1 秒时刻的日期、时间。
- 如果运行 Java 程序的本地时区是北京时区（与格林威治时间相差 8 个小时），那么上述 date1 就是 1970 年 01 月 01 日 08 时 00 分 01 秒、date2 就是 1970 年 01 月 01 日 07 时 59 分 59 秒。

- System 类的静态方法 **public long currentTimeMillis()** 获取系统当前时间。





# 8.5 Date 与 Calendar 类

## 2. Calendar 类

- **Calendar 类**在 java.util 包中。
  - 使用 Calendar 类的 static 方法 `getInstance()` 可以初始化一个日历对象，  
如：`Calendar calendar= Calendar.getInstance();`
- **calendar 对象**可以调用方法：
  - `public final void set(int year,int month,int date)`
  - `public final void set(int year,int month,int date,int hour,int minute)`
  - `public final void set(int year,int month, int date, int hour, int minute,int second)`
  - 其作用是将日历翻到任何一个时间





## 8.5 Date 与 Calendar 类

- calendar 对象可以调用方法:

- `public long getTimeInMillis()` 可以将时间表示为毫秒。
- `public final void setTime(Date date)` 使用给定的 `Date` 设置此 `Calendar` 的时间
- `public int get(int field)` : 可以获取有关年份、月份、小时、星期等信息

例如: `calendar.get(Calendar.MONTH)`; 返回一个整数, 如果该整数是 0 表示当前日历是在一月, 该整数是 1 表示当前日历是在二月等。

例如: `calendar.get(Calendar.DAY_OF_WEEK)`; 返回一个整数, 如果该整数是 1 表示星期日, 如果是 2 表示是星期一, 依次类推, 如果是 7 表示是星期六。

**例题15** 计算了 2012-9-01 和 2016-07-01 之间相隔的天数

**例题 16** Example8\_16.java  
CalendaBean.java

输出 2016 年 7 月的 “日

日	一	二	三	四	五	六
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

图 8.14 输出日历页

```
现在的时间是: 2011年1月16日 15时5分50秒  
Fri Jul 01 15:05:50 CST 2016  
与Sat Sep 01 15:05:50 CST 2012  
相隔1399天
```

图 8.13 使用 Calendar 类



# 8.5 日期的格式化

## 1 format 方法

- Formatter 类的 format 方法:

- format( 格式化模式, 日期列表 )
- 按着 “格式化模式” 返回 “日期列表” 中所列各个日期中所含数据 (年, 月, 日, 小时等数据) 的字符串表示。
- Java 已经将 format 方法做为了 String 类的静态方法, 因此, 程序可以直接使用 String 类调用 format 方法对日期进行格式化。

### 1) 格式化模式

- format 方法中的 “格式化模式” 是一个用双引号括起的字符序列 (字符串), 该字符序列中的字符由时间格式符和普通字符所构成。  
例如: " 日期 : %ty-%tm-%td "
- String s = String.format("%tY 年 %tm 月 %td 日 ", new Date(), new Date(), new Date()); 那么 s 就是 "2011 年 02 月 10 日"



## 8.6 日期的格式化

### 2) 日期列表

- `format` 方法中的“日期列表”可以用逗号分隔的 `Calendar` 对象或 `Date` 对象。
- `format` 方法默认按从左到右的顺序使用“格式化模式”中的格式符来格式“日期列表”中对应的日期，而“格式化模式”中的普通字符保留原样。

### 3) 格式化同一日期

- 可以在“格式化模式”中使用“<”，比如：“`%ty-%<tm-%<td`”中的三个格式符将格式化同一日期，即含有“<”的格式符和它前面的格式符格式同一个日期，例如：`String s = String.format("%ty 年 %<tm 月 %<td 日", new Date());` 那么 `%<tm` 和 `%<td` 都格式化 `new Date()`，因此字符串 `s` 就是：“11 年 02 月 10 日”。



## 8. 6 日期的格式化

### 2 不同区域的星期格式

- 如果想用特定地区的星期格式来表示日期中的星期，可以用 `format` 的重载方法：

**`format(Locale locale, 格式化模式, 日期列表);`**

- 其中的参数 `locale` 是一个 `Locale` 类的实例，用于表示地域。
- `Locale` 类的 `static` 常量都是 `Locale` 对象，其中 `US` 是表示美国的 `static` 常量。

比如，假设当前时间是 2011-02-10，对于（`%ta` 表示简称的星期）：

- **`String s = String.format(Locale.US,"%ta(%<tF)",new Date());`**
- 那么 `s` 是 **`"Thu(2011-02-10)"`**，
- 对于（`%tA` 表示全称的星期）
- **`String s = String.format(Locale.JAPAN,"%tA(%<tF)",new Date());`**
- 那么 `s` 是 **`"木曜日 (2011-02-10)"`**。
- 注：如果 `format` 方法不使用 `Locale` 参数格式化日期，当前应用程序所在系统的地区设置是中国，那么相当于 `locale` 参数取 `Locale.CHINA`。



# 8.7 Math、BigInteger 和 Random 类

## 1. Math 类

- Math 类在 `java.lang` 包中。Math 类包含许多用来进行科学计算的类方法，这些方法可以直接通过类名调用。另外，Math 类还有两个静态常量，它们分别是：
  - `E 2.7182828284590452354` 和 `PI 3.14159265358979323846`。
- 以下是 Math 类的常用类方法：
  - `public static long abs(double a)` 返回 a 的绝对值。
  - `public static double max(double a, double b)` 返回 a、b 的最大值。
  - `public static double min(double a, double b)` 返回 a、b 的最小值。
  - `public static double random()` 产生一个 0 到 1 之间的随机数（不包括 0 和 1）。
  - `public static double pow(double a, double b)` 返回 a 的 b 次幂。
  - `public static double sqrt(double a)` 返回 a 的平方根。
  - `public static double log(double a)` 返回 a 的对数。
  - `public static double sin(double a)` 返回正弦值。
  - `public static double asin(double a)` 返回反正弦值。



# 8.6 Math、BigInteger 和 Random 类

## 2 BigInteger 类

- java.math 包中的 BigInteger 类提供任意精度的整数运算。可以使用构造方法：
  - **public BigInteger(String val)** 构造一个十进制的 BigInteger 对象。
- 以下是 BigInteger 类的常用类方法：
  - **public BigInteger add(BigInteger val)** 返回当前大整数对象与参数指定的大整数对象的和。
  - **public BigInteger subtract(BigInteger val)** 返回当前大整数对象与参数指定的大整数对象的差。
  - **public BigInteger multiply(BigInteger val)** 返回当前大整数对象与参数指定的大整数对象的积。
  - **public BigInteger divide(BigInteger val)** 返回当前大整数对象与参数指定的大整数对象的商。
  - **public BigInteger remainder(BigInteger val)** 返回当前大整数对象与参数指定的大整数对象的余。
  - **public int compareTo(BigInteger val)** 返回当前大整数对象与参数指定的大整数的比较结果，返回值是 1、-1 或 0，分别表示当前大整数对象大于、小于或等于参数指定的大整数。
  - **public BigInteger pow(int a)** 返回当前大整数对象的 a 次幂。
  - **public String toString()** 返回当前大整数对象十进制的字符串表示。
  - **public String toString(int p)** 返回当前大整数对象 p 进制的字符串表示。
- 例题17 计算 5 的平方根以及两个大整数的和与积



# 8.7 Math、BigInteger 和 Random 类

## 3.Random 类

- **复习:** 使用 Math 类调用其类方法 random() 返回一个 0 至 1 之间的随机数 (不包括 0 和 1)。例如, 下列代码得到 1 至 100 之间的一个随机整数 (包括 1 和 100), `(int)(Math.random()*100)+1;`

### 新内容:


- Java 提供了更为灵活的用于获得随机数的 Random 类 (该类在 java.util 包中)。 Random 类的如下构造方法:
  - `public Random();`
  - `public Random(long seed);` 使用参数 seek 指定的种子创建一个 Random 对象
- 随机数生成器 random 调用不带参数的 `nextInt()` 方法:
  - `Random random=new Random();`
  - `random.nextInt();`
  - 返回一个 0 至 n 之间 (包括 0, 但不包括 n) 的随机数 随机数生成器 random 调用带参数的 `nextInt(int m)` 方法 (参数 m 必须取正整数值)

例如: `random.nextInt(100);` 返回一个 0 至 100 之间的随机整数 (包括 0, 但不包括 100)。

例如: 随机得到 true 和 false 两个表示真和假的 boolean 值,  
随机数生成器 random 调用 `nextBoolean()` 方法

例如: `random.nextBoolean();` 返回一个随机 boolean 值。





需要注意的是，对于具有相同种子的两个 **Random** 对象，二者依次调用 **nextInt()** 方法获取的随机数序列是相同的。

例子 18 演示是从 1-100 之间随机得到 6 个不同的数

**Example8\_18.java**

得到随机数类 **GetRandomNumber**





## 8.8 数字格式化

### 1 Formatter 类

Formatter 类提供了一个和 C 语言 printf 函数类似的 format 方法：

**format (格式化模式, 值列表)**

该方法按着“**格式化模式**”返回“**值列表**”的字符串表示。

Java 已经将 format 方法做为了 String 类的静态方法，因此，程序可以直接使用 String 类调用 format 方法对数字进行格式化。

**1) 格式化模式** format 方法中的“**格式化模式**”是一个用双引号括起的字符序列 ( 字符串 ), 该字符序列中的字符由格式符和普通字符所构成。

例如: “**输出结果 %d,%f,%d**” 中的 %d 和 %f 是格式符号。

format 方法返回的字符串就是“格式化模式”中的格式符被替换为它得到的格式化结果后的字符串。

例如: **String s = String.format(“%.2f”,3.141592);** 那么 s 就是 “3.14”。

**2) 值列表** format 方法中的“值列表”是用逗号分隔的变量、常量或表达式。 例如:

**String s=format(“%d 元 %0.3f 公斤 %d 台 ”,888,999.777666,123);** 那么 s 就是 “888 元 999.778 公斤 123 台”。



## 8.8 数字格式化

### 3) 格式化顺序

`format` 方法默认按从左到右的顺序使用“格式化模式”中的格式符来格式化“值列表”中对应的值，而“格式化模式”中的普通字符保留原样。

例如，假设 `int` 型变量 `x` 和 `double` 型变量 `y` 的值分别是 888 和 3.1415926，

那么对于 `String s = format(" 从左向右: %d,%.3f,%d",x,y,100);`

字符串 `s` 就是：从左向右： 888,3.142,100

**注1**：改变默认的顺序（从左向右）进行格式化，在格式符前面添加索引符号 `index$`，例如，`1$` 表示“值列表”中的第 1 个，`2$` 表示“值列表”中的第 2 个，

对于：`String s=String.format(" 不是从左向右: %2$.3f,%3$d,%1$d",x,y,100);`

字符串 `s` 就是 不是从左向右： 3.142,100,888

**注2**：如果准备在“格式化模式”中包含普通的 `%`，在编写代码时需要连续键入两个 `%`，如：

`String s=String.format("%d%%",89);` 输字符串 `s` 是： "89%"



# 8.8 数字格式化

## 2 格式化整数

1 ) **%d** , **%o** , **%x** 和 **%** 格式符可格式化 byte 、 Byte 、 short 、 Short 、 int 、 Integer 、 long 和 Long 型数据, 详细说明见 Page204. 例如, 对于:

String s

```
= String.format("%d,%o,%x,%X",703576,703576,703576,703576);
```

字符串 s 就是: 703576,2536130,abc58,ABC58

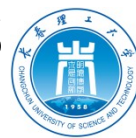
## 2 ) 修饰符

加号修饰符 "+" 格式化正整数时, 强制添加上正号, 例如, % +d 将 123 格式化为 "+123" 。

逗号修饰符 “,” 格式化整数时, 按 "千" 分组, 例如, 对于:

```
String s=String.format("按千分组 :%,d 。按千分组带正号 %  
+,d",1235678,9876);
```

字符串 s 就是: 按千分组 :1,235,678 。按千分组带正号 +9,876



### 3) 数据的宽度

数据的宽度就是 `format` 方法返回的字符串的长度。规定数据宽度的一般格式为 “`%md`”, 其效果是在数字的左面增加空格; 或 “`%-md`”, 其效果是在数字的右面增加空格, 例如, 将数字 59 格式化为宽度为 8 的字符串:

```
String s=String.format("%8d",59);
```

字符串 `s` 就是: “ 59”, 其长度 (`s.length()`) 为 8, 即 `s` 在 59 左面添加了 6 个空格字符。

对于: `String s=String.format("%-8d",59);` 字符串 `s` 就是: “59”, 其长度 (`s.length()`) 为 8。

对于: `String s=String.format("%5d%5d%8d",59,60,90);` 字符串 `s` 就是: “ 59 60 90”(长度为 18)。

注: 如果实际数字的宽度大于格式中指定的宽度, 就按数字的实际宽度进行格式化。

可以在宽度的前面增加前缀 0, 表示用数字 0 (不用空格) 来填充宽度左面的富裕部分, 例如: `String s=String.format("%08d",12);` 字符串 `s` 就是: “00000012”, 其长度 (`s.length()`) 为 8, 即 `s` 在 12 的左面添加了 6 个数字 0。



## 8.8 数字格式化

### 3 格式化浮点数 \_1

1) **%f,%e(%E),%g(%G) 和 %a(%A)** 格式符可格式化 float、Float、double 和 Double:

**%f** 将值格式化为十进制浮点数，小数保留 6 位。

**%e** ( **%E** ) 将值格式化为科学记数法的十进制的浮点数 ( **%E** 在格式化时将其中的指数符号大写，例如 5E10 )。

例如，对于：`String s = String.format("%f,%e",13579.98,13579.98);`

字符串 s 就是：

13579.980000,1.357998e+04

### 2) 修饰符

加号修饰符 "+" 格式化正数时，强制添加上正号，例如 **%+f** 将 123.78 格式化为 "+123.78"，**%+E** 将 123.78 格式化为 "+1.2378E+2"。

逗号修饰符 “,” 格式化浮点数时，将整数部分按“千”分组，

例如，对于：`String s=String.format("%+f",1235678.9876);`



## 8.8 数字格式化

### 3) 限制小数位数与数据的“宽度”

"%.nf" 可以限制小数的位数, 其中的 n 是保留的小数位数, 例如 `%.3f` 将 6.1256 格式化为 "6.126" (保留 3 位小数)。

规定宽度的一般格式为 “%mf”(在数字的左面增加空格), 或 “%-md”(在数字的右面增加空格)。

例如, 将数字 59.88 格式化为宽度为 11 的字符串

`String s=String.format("%11f",59.88);` 字符串 s 就是: “ 59.880000”, 其长度 (s.length()) 为 11

`String s=String.format("%-11f",59.88);` 字符串 s 就是: “59.880000 ”, 其长度 (s.length()) 为 11

➤在指定宽度的同时也可以限制小数位数 (%m.nf),

`String s=String.format("%11.2f",59.88);` 字符串 s 就是: “     59.88”, 即 s 在 59.88 左面添加了 6 个空格字符。

➤在宽度的前面增加前缀 0, 表示用数字 0 (不用空格) 来填充宽度左面的富裕部分, 例如:

`String s=String.format("%011f",59.88);` 字符串 s 就是: “0059.880000”, 其长度 (s.length()) 为 11。

**注:** 如果实际数字的宽度大于格式中指定的宽度, 就按数字的实际宽度进行格式化



## 8.9 Class 类与 Console 类

### 8.9.1 Class 类

1. 使用 Class 的类方法得到一个和某类（参数 className 指定的类）相关的 Class 对象，

**public static Class forName(String className) throws ClassNotFoundException**

上述方法返回一个和参数 className 指定的类相关的 Class 对象。如果类在某个包中，className 必须带有包名，例如，className="java.util.Date"。

2. 步骤 1 中获得的 Class 对象调用

**public Object newInstance() throws Exception,**

方法就可以得到一个 className 类的对象

要特别注意的是：使用 Class 对象调用 newInstance() 实例化一个 className 类的对象时，className 类必须有无参数的构造方法

例子 20 使用 Class 对象得到一个 Rect 类以及 java.util 包中 Date 类的对象。运行效果如图 8.23 所示

```
rect的面积20000.0  
2016-10-02 09:15:53 星期日
```

图 8.23 用 Class 实例化对象





## 8.9.2 Console 类

如果希望在键盘输入一行文本，但不想让该文本回显，即不在命令行显示，那么就需要使用 `java.io` 包中的 `Console` 类的对象来完成。首先使用 `System` 类调用 `console()` 方法返回一个 `Console` 类的一个对象，比如 `cons`：

```
Console cons = System.console();
```

然后，`cons` 调用 `readPassword()` 方法读取用户在键盘输入的一行文本，并将文本以一个 `char` 数组返回：

```
char[] passwd = cons.readPassword();
```

例子 21 中，模拟用户用户输入的密码，如果输入正确（`I love this game`），那么，程序让用户看到“你好，欢迎你！”。程序允许用户 2 次输入的密码不正确，一旦超过 2 次，程序将立刻退出。





# 8.10 Pattern 与 Matcher 类\_1

- Java 提供了专门用来进行模式匹配的 Pattern 类和 Matcher 类，这些类在 `java.util.regex` 包中。
- 以下结合具体问题来讲解使用 Pattern 类和 Matcher 类的步骤。假设有字符串：

`String input = "hello,good morning,this is a good idea"`

- 我们想知道 input 从哪个位置开始至哪个位置结束曾出现了字符串 good 。
- 使用 Pattern 类和 Matcher 类检索字符串 str 中的子字符串的步骤如下：

1. 建立模式对象
2. 得到匹配对象



# 8.10 Pattern 与 Matcher 类\_1

## 1. 建立模式对象

- 使用正则表达式 `regex` 做参数得到一个称为模式的 `Pattern` 类的实例 `pattern` :

例如: `String regex = "good";`

`Pattern pattern = Pattern.compile(regex);`

- 模式对象是对正则表达式的封装。 `Pattern` 类调用类方法 `compile(String regex)` 返回一个模式对象, 其中的参数 `regex` 是一个正则表达式, 称为模式对象使用的模式。



# 8. 10 Pattern 与 Matcher 类 \_2

## 2. 得到匹配对象

- 模式对象 `pattern` 调用 `matcher(CharSequence input)` 方法返回一个 `Matcher` 对象 `matcher`，称为匹配对象

**`Matcher matcher = pattern.matcher(input);`**

- `Matcher` 对象 `matcher` 可以使用下列方法寻找字符串 `input` 中是否有和模式 `regex` 匹配的子序列（`regex` 是创建模式对象 `pattern` 时使用的正则表达式）



## 8.10 Pattern 与 Matcher 类\_2

- **public boolean find()** : 寻找 input 和 regex 匹配的下一子序列，如果成功该方法返回 true，否则返回 false
- **public boolean matches()** : matcher 调用该方法判断 input 是否完全和 regex 匹配
- **public boolean lookingAt()** : matcher 调用该方法判断从 input 的开始位置是否有和 regex 匹配的子序列
- **public boolean find(int start)** : matcher 调用该方法判断 input 从参数 start 指定位置开始是否有和 regex 匹配的子序列




## 8. 10 Pattern 与 Matcher 类 \_2

□ **public String replaceAll(String replacement)** matcher 调用该方法可以返回一个字符串，该字符串是通过把 input 中与模式 regex 匹配的子字符串全部替换为参数 replacement 指定的字符串得到的

□ **public String replaceFirst(String replacement)** matcher 调用该方法可以返回一个字符串，该字符串是通过把 input 中第 1 个与模式 regex 匹配的子字符串替换为参数 replacement 指定的字符串得到的





## 例子 22 计算了一个账单的总价格

```
import java.util.regex.*;
public class Example8_22 {
    public static void main(String args[ ]) {
        String s = " 市话 76.8 元 , 长途 :167.38 元 , 短信 12.68";
        String regex = "[0123456789.]+"; // 匹配数字序列
        Pattern p = Pattern.compile(regex); // 模式对象
        Matcher m = p.matcher(s);          // 匹配对象
        double sum = 0;
        while(m.find()) {
            String item = m.group();
            System.out.println(item);
            sum = sum+Double.parseDouble(item);
        }
        System.out.println(" 账单总价格 :"+sum);
    }
}
```



## 8.11 应用举例

本节用 Java 程序模拟抢红包，这里给出的随机抢红包算法比较简单，比如，假设当前是 **5.2** 圆，参与抢红包的人是 **6** 人。那么第一个人抢到的金额  $m$  是一个在 **0-519** 之间的随机数（按分表示钱的金额），如果  $m$  是 0，需要把  $m$  赋值成 1（保证用户至少能抢到 1 分钱），如果  $m$  不是 0，那么 **520- $m$**  是**剩余的金额**，要求剩余的金额必须保证其余 5 个人都至少能抢到 1 分钱，否则  $m$  要减去多抢到的金额。读者可以阅读代码，理解类以及其中方法。

例子 23 中有 2 个重要的类 **RedEnvelope** 以及它的子类 **RandomRedEnvelope**。**RedEnvelope** 类是抽象类，规定了子类必须要重写的抢红包的方法 **giveMoney()**。子类 **RandomRedEnvelope** 重写 **giveMoney()** 方法实现随机抢红包（随机红包），效果如图 8.24

```
以下用循环输出6个人抢5.20圆的随机红包:  
3.64    0.67    0.43    0.44    0.01    0.01  
5.20圆的红包被抢完
```

图 8.24 抢红包

