Java 日期时间

java.util 包提供了 Date 类来封装当前的日期和时间。 Date 类提供两个构造函数来实例化 Date 对象。

第一个构造函数使用当前日期和时间来初始化对象。

```
Date( )
```

第二个构造函数接收一个参数，该参数是从1970年1月1日起的毫秒数。

```
Date(long millisec)
```

Date对象创建以后，可以调用下面的方法。

序号	方法和描述
1	**boolean after(Date date)** 若当调用此方法的Date对象在指定日期之后返回true,否则返回false。
2	**boolean before(Date date)** 若当调用此方法的Date对象在指定日期之前返回true,否则返回false。
3	**Object clone( )** 返回此对象的副本。
4	**int compareTo(Date date)** 比较当调用此方法的Date对象和指定日期。两者相等时候返回0。调用对象在指定日期之前则返回负数。调用对象在指定日期之后则返回正数。
5	**int compareTo(Object obj)** 若obj是Date类型则操作等同于compareTo(Date) 。否则它抛出ClassCastException。
6	**boolean equals(Object date)** 当调用此方法的Date对象和指定日期相等时候返回true,否则返回false。
7	**long 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。
8	**int hashCode( )** 返回此对象的哈希码值。
9	**void setTime(long time)** 用自1970年1月1日00:00:00 GMT以后time毫秒数设置时间和日期。
10	**String 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)。

获取当前日期时间

Java中获取当前日期和时间很简单，使用 Date 对象的 toString() 方法来打印当前日期和时间，如下所示：

实例

```
import java.util.Date;

public class DateDemo {
   public static void main(String args[]) {
       // 初始化 Date 对象
       Date date = new Date();

       // 使用 toString() 函数显示日期时间
       System.out.println(date.toString());
   }
}
```

运行实例 »

以上实例编译运行结果如下:

```
Mon May 04 09:51:52 CDT 2013
```

日期比较

Java使用以下三种方法来比较两个日期:

使用 getTime() 方法获取两个日期（自1970年1月1日经历的毫秒数值），然后比较这两个值。

使用方法 before()，after() 和 equals()。例如，一个月的12号比18号早，则 new Date(99, 2, 12).before(new Date (99, 2, 18)) 返回true。

使用 compareTo() 方法，它是由 Comparable 接口定义的，Date 类实现了这个接口。

使用 SimpleDateFormat 格式化日期

SimpleDateFormat 是一个以语言环境敏感的方式来格式化和分析日期的类。SimpleDateFormat 允许你选择任何用户自定义日期时间格式来运行。例如:

实例

```
import  java.util.*;
import java.text.*;

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

      Date dNow = new Date( );
      SimpleDateFormat ft = new SimpleDateFormat ("yyyy-MM-dd hh:
mm:ss");

      System.out.println("当前时间为: " + ft.format(dNow));
   }
}
```

运行实例 »

```
SimpleDateFormat ft = new SimpleDateFormat ("yyyy-MM-dd hh:mm:ss");
```

这一行代码确立了转换的格式，其中 yyyy 是完整的公元年，MM 是月份，dd 是日期，HH:mm:ss 是时、分、秒。

注意:有的格式大写，有的格式小写，例如 MM 是月份，mm 是分；HH 是 24 小时制，而 hh 是 12 小时制。

以上实例编译运行结果如下:

```
当前时间为: 2018-09-06 10:16:34
```

日期和时间的格式化编码

时间模式字符串用来指定时间格式。在此模式中，所有的 ASCII 字母被保留为模式字母，定义如下：

字母	描述	示例
G	纪元标记	AD
y	四位年份	2001
M	月份	July or 07
d	一个月的日期	10
h	A.M./P.M. (1~12)格式小时	12
H	一天中的小时 (0~23)	22
m	分钟数	30
s	秒数	55
S	毫秒数	234
E	星期几	Tuesday
D	一年中的日子	360
F	一个月中第几周的周几	2 (second Wed. in July)
w	一年中第几周	40
W	一个月中第几周	1
a	A.M./P.M. 标记	PM
k	一天中的小时(1~24)	24
K	A.M./P.M. (0~11)格式小时	10
z	时区	Eastern Standard Time
'	文字定界符	Delimiter
"	单引号	`

使用printf格式化日期

printf 方法可以很轻松地格式化时间和日期。使用两个字母格式，它以 %t 开头并且以下面表格中的一个字母结尾。

转 换 符	说　明	示　例
c	包括全部日期和时间信息	星期六 十月 27 14:21:20 CST 2007
F	"年-月-日"格式	2007-10-27

D	"月/日/年"格式	10/27/07
r	"HH:MM:SS PM"格式（12时制）	02:25:51 下午
T	"HH:MM:SS"格式（24时制）	14:28:16
R	"HH:MM"格式（24时制）	14:28

更多 **printf** 解析可以参见：Java 格式化输出 printf 例子

实例

实例

```
import java.util.Date;

public class DateDemo {

  public static void main(String args[]) {
     // 初始化 Date 对象
     Date date = new Date();

     //c的使用
    System.out.printf("全部日期和时间信息：%tc%n",date);
    //f的使用
    System.out.printf("年-月-日格式：%tF%n",date);
    //d的使用
    System.out.printf("月/日/年格式：%tD%n",date);
    //r的使用
    System.out.printf("HH:MM:SS PM格式（12时制）：%tr%n",date);
    //t的使用
    System.out.printf("HH:MM:SS格式（24时制）：%tT%n",date);
    //R的使用
    System.out.printf("HH:MM格式（24时制）：%tR",date);
  }
}
```

以上实例编译运行结果如下:

```
全部日期和时间信息：星期一 九月 10 10:43:36 CST 2012
年-月-日格式：2012-09-10
月/日/年格式：09/10/12
HH:MM:SS PM格式（12时制）：10:43:36 上午
HH:MM:SS格式（24时制）：10:43:36
HH:MM格式（24时制）：10:43
```

如果你需要重复提供日期，那么利用这种方式来格式化它的每一部分就有点复杂了。因此，可以利用一个格式化字符串指出要被格式化的参数的索引。

索引必须紧跟在%后面，而且必须以$结束。例如:

实例

```
import java.util.Date;

public class DateDemo {

   public static void main(String args[]) {
       // 初始化 Date 对象
       Date date = new Date();

       // 使用toString()显示日期和时间
       System.out.printf("%1$s %2$tB %2$td, %2$tY",
```

```
                        "Due date:", date);
   }
}
```

运行实例 »

以上实例编译运行结果如下:

```
Due date: February 09, 2014
```

或者，你可以使用 < 标志。它表明先前被格式化的参数要被再次使用。例如:

实例

```
import java.util.Date;

public class DateDemo {

   public static void main(String args[]) {
       // 初始化 Date 对象
       Date date = new Date();

       // 显示格式化时间
       System.out.printf("%s %tB %<te, %<tY",
                         "Due date:", date);
   }
}
```

运行实例 »

以上实例编译运行结果如下:

```
Due date: February 09, 2014
```

定义日期格式的转换符可以使日期通过指定的转换符生成新字符串。这些日期转换符如下所示:

实例

```
import java.util.*;

public class DateDemo {
   public static void main(String args[]) {
       Date date=new Date();
        //b的使用，月份简称
        String str=String.format(Locale.US,"英文月份简称：%tb",dat
e);
        System.out.println(str);
        System.out.printf("本地月份简称：%tb%n",date);
        //B的使用，月份全称
        str=String.format(Locale.US,"英文月份全称：%tB",date);
        System.out.println(str);
        System.out.printf("本地月份全称：%tB%n",date);
        //a的使用，星期简称
        str=String.format(Locale.US,"英文星期的简称：%ta",date);
        System.out.println(str);
        //A的使用，星期全称
        System.out.printf("本地星期的简称：%tA%n",date);
        //C的使用，年前两位
        System.out.printf("年的前两位数字（不足两位前面补0）：%tC%n"
,date);
        //y的使用，年后两位
        System.out.printf("年的后两位数字（不足两位前面补0）：%ty%n"
,date);
```

```
        //j的使用，一年的天数
        System.out.printf("一年中的天数（即年的第几天）：%tj%n",dat
e);
        //m的使用，月份
        System.out.printf("两位数字的月份（不足两位前面补0）：%tm%n"
,date);
        //d的使用，日（二位，不够补零）
        System.out.printf("两位数字的日（不足两位前面补0）：%td%n",
date);
        //e的使用，日（一位不补零）
        System.out.printf("月份的日（前面不补0）：%te",date);
    }
}
```

输出结果为：

```
英文月份简称：May
本地月份简称：五月
英文月份全称：May
本地月份全称：五月
英文星期的简称：Thu
本地星期的简称：星期四
年的前两位数字（不足两位前面补0）：20
年的后两位数字（不足两位前面补0）：17
一年中的天数（即年的第几天）：124
两位数字的月份（不足两位前面补0）：05
两位数字的日（不足两位前面补0）：04
月份的日（前面不补0）：4
```

解析字符串为时间

SimpleDateFormat 类有一些附加的方法，特别是parse()，它试图按照给定的SimpleDateFormat 对象的格式化存储来解析字符串。例如：

实例

```
import java.util.*;
import java.text.*;

public class DateDemo {

   public static void main(String args[]) {
      SimpleDateFormat ft = new SimpleDateFormat ("yyyy-MM-dd");

      String input = args.length == 0 ? "1818-11-11" : args[0];

      System.out.print(input + " Parses as ");

      Date t;

      try {
          t = ft.parse(input);
          System.out.println(t);
      } catch (ParseException e) {
          System.out.println("Unparseable using " + ft);
      }
   }
}
```

运行实例 »

以上实例编译运行结果如下：

```
$ java DateDemo
1818-11-11 Parses as Wed Nov 11 00:00:00 GMT 1818
$ java DateDemo 2007-12-01
2007-12-01 Parses as Sat Dec 01 00:00:00 GMT 2007
```

Java 休眠(sleep)

sleep()使当前线程进入停滞状态（阻塞当前线程），让出CPU的使用、目的是不让当前线程独自霸占该进程所获的CPU资源，以留一定时间给其他线程执行的机会。

你可以让程序休眠一毫秒的时间或者到您的计算机的寿命长的任意段时间。例如，下面的程序会休眠3秒：

实例

```
import java.util.*;

public class SleepDemo {
   public static void main(String args[]) {
      try {
         System.out.println(new Date( ) + "\n");
         Thread.sleep(1000*3);   // 休眠3秒
         System.out.println(new Date( ) + "\n");
      } catch (Exception e) {
          System.out.println("Got an exception!");
      }
   }
}
```

运行实例 »

以上实例编译运行结果如下:

```
Thu Sep 17 10:20:30 CST 2015

Thu Sep 17 10:20:33 CST 2015
```

测量时间

下面的一个例子表明如何测量时间间隔（以毫秒为单位）：

实例

```
import java.util.*;

public class DiffDemo {

   public static void main(String args[]) {
      try {
         long start = System.currentTimeMillis( );
         System.out.println(new Date( ) + "\n");
         Thread.sleep(5*60*10);
         System.out.println(new Date( ) + "\n");
         long end = System.currentTimeMillis( );
         long diff = end - start;
         System.out.println("Difference is : " + diff);
      } catch (Exception e) {
         System.out.println("Got an exception!");
      }
   }
}
```

运行实例 »

以上实例编译运行结果如下:

```
Fri Jan 08 09:48:47 CST 2016

Fri Jan 08 09:48:50 CST 2016

Difference is : 3019
```

Calendar类

我们现在已经能够格式化并创建一个日期对象了，但是我们如何才能设置和获取日期数据的特定部分呢，比如说小时，日，或者分钟? 我们又如何在日期的这些部分加上或者减去值呢? 答案是使用Calendar 类。

Calendar类的功能要比Date类强大很多，而且在实现方式上也比Date类要复杂一些。

Calendar类是一个抽象类，在实际使用时实现特定的子类的对象，创建对象的过程对程序员来说是透明的，只需要使用getInstance方法创建即可。

创建一个代表系统当前日期的Calendar对象

```
Calendar c = Calendar.getInstance();//默认是当前日期
```

创建一个指定日期的Calendar对象

使用Calendar类代表特定的时间，需要首先创建一个Calendar的对象，然后再设定该对象中的年月日参数来完成。

```
//创建一个代表2009年6月12日的Calendar对象
Calendar c1 = Calendar.getInstance();
c1.set(2009, 6 - 1, 12);
```

Calendar类对象字段类型

Calendar类中用以下这些常量表示不同的意义，jdk内的很多类其实都是采用的这种思想

常量	描述
Calendar.YEAR	年份
Calendar.MONTH	月份
Calendar.DATE	日期
Calendar.DAY_OF_MONTH	日期，和上面的字段意义完全相同
Calendar.HOUR	12小时制的小时
Calendar.HOUR_OF_DAY	24小时制的小时
Calendar.MINUTE	分钟
Calendar.SECOND	秒
Calendar.DAY_OF_WEEK	星期几

Calendar类对象信息的设置

Set设置

如：

```
Calendar c1 = Calendar.getInstance();
```

调用：

```
public final void set(int year,int month,int date)
```

```
c1.set(2009, 6, 12);//把Calendar对象c1的年月日分别设这为：2009、6、12
```

利用字段类型设置

如果只设定某个字段，例如日期的值，则可以使用如下set方法：

```
public void set(int field,int value)
```

把 c1对象代表的日期设置为10号，其它所有的数值会被重新计算

```
c1.set(Calendar.DATE,10);
```

把c1对象代表的年份设置为2008年，其他的所有数值会被重新计算

```
c1.set(Calendar.YEAR,2008);
```

其他字段属性set的意义以此类推

Add设置

```
Calendar c1 = Calendar.getInstance();
```

把c1对象的日期加上10，也就是c1也就表示为10天后的日期，其它所有的数值会被重新计算

```
c1.add(Calendar.DATE, 10);
```

把c1对象的日期减去10，也就是c1也就表示为10天前的日期，其它所有的数值会被重新计算

```
c1.add(Calendar.DATE, -10);
```

其他字段属性的add的意义以此类推

Calendar类对象信息的获得

```
Calendar c1 = Calendar.getInstance();
// 获得年份
int year = c1.get(Calendar.YEAR);
// 获得月份
int month = c1.get(Calendar.MONTH) + 1;
// 获得日期
int date = c1.get(Calendar.DATE);
// 获得小时
```

```
int hour = c1.get(Calendar.HOUR_OF_DAY);
// 获得分钟
int minute = c1.get(Calendar.MINUTE);
// 获得秒
int second = c1.get(Calendar.SECOND);
// 获得星期几（注意（这个与Date类是不同的）：1代表星期日、2代表星期1、3代表星期二，以此类推）
int day = c1.get(Calendar.DAY_OF_WEEK);
```

GregorianCalendar类

Calendar类实现了公历日历，GregorianCalendar是Calendar类的一个具体实现。

Calendar 的getInstance（）方法返回一个默认用当前的语言环境和时区初始化的GregorianCalendar对象。GregorianCalendar定义了两个字段：AD和BC。这是代表公历定义的两个时代。

下面列出GregorianCalendar对象的几个构造方法：

序号	构造函数和说明
1	**GregorianCalendar()** 在具有默认语言环境的默认时区内使用当前时间构造一个默认的 GregorianCalendar。
2	**GregorianCalendar(int year, int month, int date)** 在具有默认语言环境的默认时区内构造一个带有给定日期设置的 GregorianCalendar
3	**GregorianCalendar(int year, int month, int date, int hour, int minute)** 为具有默认语言环境的默认时区构造一个具有给定日期和时间设置的 GregorianCalendar。
4	**GregorianCalendar(int year, int month, int date, int hour, int minute, int second)** 为具有默认语言环境的默认时区构造一个具有给定日期和时间设置的 GregorianCalendar。
5	**GregorianCalendar(Locale aLocale)** 在具有给定语言环境的默认时区内构造一个基于当前时间的 GregorianCalendar。
6	**GregorianCalendar(TimeZone zone)** 在具有默认语言环境的给定时区内构造一个基于当前时间的 GregorianCalendar。
7	**GregorianCalendar(TimeZone zone, Locale aLocale)** 在具有给定语言环境的给定时区内构造一个基于当前时间的 GregorianCalendar。

这里是GregorianCalendar 类提供的一些有用的方法列表：

序号	方法和说明
1	**void add(int field, int amount)** 根据日历规则，将指定的（有符号的）时间量添加到给定的日历字段中。

2	**protected void computeFields()** 转换UTC毫秒值为时间域值
3	**protected void computeTime()** 覆盖Calendar ， 转换时间域值为UTC毫秒值
4	**boolean equals(Object obj)** 比较此 GregorianCalendar 与指定的 Object。
5	**int get(int field)** 获取指定字段的时间值
6	**int getActualMaximum(int field)** 返回当前日期，给定字段的最大值
7	**int getActualMinimum(int field)** 返回当前日期，给定字段的最小值
8	**int getGreatestMinimum(int field)** 返回此 GregorianCalendar 实例给定日历字段的最高的最小值。
9	**Date getGregorianChange()** 获得格里高利历的更改日期。
10	**int getLeastMaximum(int field)** 返回此 GregorianCalendar 实例给定日历字段的最低的最大值
11	**int getMaximum(int field)** 返回此 GregorianCalendar 实例的给定日历字段的最大值。
12	**Date getTime()** 获取日历当前时间。
13	**long getTimeInMillis()** 获取用长整型表示的日历的当前时间
14	**TimeZone getTimeZone()** 获取时区。
15	**int getMinimum(int field)** 返回给定字段的最小值。
16	**int hashCode()** 重写hashCode.
17	**boolean isLeapYear(int year)** 确定给定的年份是否为闰年。
18	**void roll(int field, boolean up)** 在给定的时间字段上添加或减去（上/下）单个时间单元，不更改更大的字段。
19	**void set(int field, int value)**

	用给定的值设置时间字段。
20	**void set(int year, int month, int date)** 设置年、月、日的值。
21	**void set(int year, int month, int date, int hour, int minute)** 设置年、月、日、小时、分钟的值。
22	**void set(int year, int month, int date, int hour, int minute, int second)** 设置年、月、日、小时、分钟、秒的值。
23	**void setGregorianChange(Date date)** 设置 GregorianCalendar 的更改日期。
24	**void setTime(Date date)** 用给定的日期设置Calendar的当前时间。
25	**void setTimeInMillis(long millis)** 用给定的long型毫秒数设置Calendar的当前时间。
26	**void setTimeZone(TimeZone value)** 用给定时区值设置当前时区。
27	**String toString()** 返回代表日历的字符串。

实例

实例

```
import java.util.*;

public class GregorianCalendarDemo {

   public static void main(String args[]) {
      String months[] = {
      "Jan", "Feb", "Mar", "Apr",
      "May", "Jun", "Jul", "Aug",
      "Sep", "Oct", "Nov", "Dec"};

      int year;
      // 初始化 Gregorian 日历
      // 使用当前时间和日期
      // 默认为本地时间和时区
      GregorianCalendar gcalendar = new GregorianCalendar();
      // 显示当前时间和日期的信息
      System.out.print("Date: ");
      System.out.print(months[gcalendar.get(Calendar.MONTH)]);
      System.out.print(" " + gcalendar.get(Calendar.DATE) + " ");
      System.out.println(year = gcalendar.get(Calendar.YEAR));
      System.out.print("Time: ");
      System.out.print(gcalendar.get(Calendar.HOUR) + ":");
      System.out.print(gcalendar.get(Calendar.MINUTE) + ":");
      System.out.println(gcalendar.get(Calendar.SECOND));

      // 测试当前年份是否为闰年
      if(gcalendar.isLeapYear(year)) {
         System.out.println("当前年份是闰年");
      }
      else {
```

```
            System.out.println("当前年份不是闰年");
        }
    }
}
```

运行实例 »

以上实例编译运行结果如下：

```
Date: Apr 22 2009
Time: 11:25:27
当前年份不是闰年
```