Java中Date和Calender类的使用方法

2009-10-04 20:49

|  |  |
| --- | --- |
| |  | | --- | | Date和Calendar是Java类库里提供对时间进行处理的类，由于日期在商业逻辑的应用中占据着很重要的地位，所以在这里想对这两个类进行一个基本的讲解，由于技术有限，不到之处请指正。  Date类顾名思义，一看就知道是和日期有关的类了，这个类最主要的作用就是获得当前时间了，然而这个类里面也具有设置时间以及一些其他的功能，可是由于 本身设计的问题，这些方法却遭到众多批评，而这些遭受批评的功能都已移植到另外一个类里面，这就是今天要讲到的第二个类Calendar里面。  在讲两个类之前，这里又不能不多提一个类，那就是DateFormat类，这个类是用来格式化日期的，稍后也会讲到。  首先，让我们来看一个获取当前时间的例子：  Date date = new Date(); System.out.println(date.getTime());上面的语句首先创建了Date的一个对象，接着使用getTime方法获得当前 的时间，但是注意了，输出后的结果确实一串长整型的数字，这是为什么？实际上这是系统根据当前时间计算出来的一个long型的数，至于是如何计算出来的就 不在本文中讲述了，那既然这样的话又如何显示正确的时间呢？这就要利用到上面的DateFormat类了，这个类是一个基类，它有一个子类是 SimpleDateFormat，具体用法请看下面的代码：  Date date = new Date(); SimpleDateFormat dateFm = new SimpleDateFormat("EEEE-MMMM-dd-yyyy"); System.out.println(dateFm.format(date));这段代码开始创建了一个Date的对象，用来获取当前时间，而重点就 在于后面的SimpleDateFormat对象，这个对继承了DateFormat，利用format方法对Date对象进行格式化，然后输出，而格式 的定制是由用户定制的，EEEE代表星期，MMMM代表月份，而dd代表日，yyyy代表年。使用这个方法就可以根据用户自定义的格式进行输出时间。  上面介绍了由用户自定义格式的输出时间，下面将来介绍通过JAVA类库提供的标准格式输出时间，这就要用到DateFormat类了，请看以下代码：  Date date = new Date(); DateFormat dateFm = DateFormat.getDateTimeInstance(DateFormat.SHORT, DateFormat.SHORT); System.out.println(dateFm.format(date));这里使用的方法和用户自定义的方法差不多，只是这里使用的是一个抽象 类，由于DateFormat是一个抽象类，所以它不能通过构造函数构造对象，在这里是通过getDateTimeInstance()方法获得该对象， 而所传递的参数就是DateFormat里面定义的一些常量，系统根据这些常量输出当前时间，由于这里使用的是getDateTimeInstance方 法，所以将传递两个常量参数，用来分别格式化日期和当前的时间。  上面讲述了如何获得系统时间以及如何格式化输出，那如果想获取或者设置时间当中的某一部分又该如何呢？例如年，月，日。这就要靠Calendar这个类 了，这个类也是一个抽象类，它有一个子类GregorianCalendar，接下来我会利用这个子类来演示这个过程，请看以下代码：  DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.FULL); GregorianCalendar cal = new GregorianCalendar(); cal.setTime(new Date()); System.out.println("System Date: " + dateFormat.format(cal.getTime())); cal.set(GregorianCalendar.DAY\_OF\_WEEK,GregorianCalendar.FRIDAY); System.out.println("After Setting Day of Week to Friday: " + dateFormat.format(cal.getTime())); 这段代码当中，首先创建了一个DateFormat对象进行格式设置，接着创建了一个GregorianCalendar对象cal，接着使用 cal.setTime()方法设置cal对象中的时间为当前时间，然后通过format格式化由cal.getTime()返回的时间进行输出，后面利 用set方法设置cal的日期为当前星期的FRIDAY，此时cal中存储的时间就是这个星期五的该时刻，而后面利用format格式化输出，假如当前时 间为2005年1月27日星期4的11点30分，那么最后将那句将会输出2005年1月28日星期5的11点30分。  在JDK 1.1后，java.util.Date 类型的大多数方法已经不推荐使用了。代替它的是Calendar。  而在java.sql.Date 和 java.util.Date之间，有些微妙的关系。  如何将Date类型插入数据库中，成为好多人的一个不小的障碍。  我们将讨论下面的类:  1、具体类(和抽象类相对)java.util.Date  2、抽象类java.text.DateFormat 和它的一个具体子类,java.text.SimpleDateFormat  3、抽象类java.util.Calendar 和它的一个具体子类,java.util.GregorianCalendar  具体类可以被实例化, 但是抽象类却不能. 你首先必须实现抽象类的一个具体子类.  Date 类从Java 开发包(JDK) 1.0 就开始进化, 当时它只包含了几个取得或者设置一个日期数据的各个部分的方法, 比如说月, 日, 和年. 这些方法现在遭到了批评并且已经被转移到了Calendar类里去了, 我们将在本文中进一步讨论它. 这种改进旨在更好的处理日期数据的国际化格式. 就象在JDK 1.1中一样, Date 类实际上只是一个包裹类, 它包含的是一个长整型数据, 表示的是从GMT(格林尼治标准时间)1970年, 1 月 1日00:00:00这一刻之前或者是之后经历的毫秒数.  一、创建一个日期对象r  让我们看一个使用系统的当前日期和时间创建一个日期对象并返回一个长整数的简单例子. 这个时间通常被称为Java 虚拟机(JVM)主机环境的系统时间.  import java.util.Date;  public class DateExample1 {  public static void main(String[] args) {  // Get the system date/time  Date date = new Date();  System.out.println(date.getTime());  }  }  在星期六, 2001年9月29日, 下午大约是6:50的样子, 上面的例子在系统输出设备上显示的结果是 1001803809710. 在这个例子中,值得注意的是我们使用了Date 构造函数创建一个日期对象, 这个构造函数没有接受任何参数. 而这个构造函数在内部使用了System.currentTimeMillis() 方法来从系统获取日期.  那么, 现在我们已经知道了如何获取从1970年1月1日开始经历的毫秒数了. 我们如何才能以一种用户明白的格式来显示这个日期呢? 在这里类java.text.SimpleDateFormat 和它的抽象基类 java.text.DateFormat 就派得上用场了.  二、日期数据的定制格式  假如我们希望定制日期数据的格式, 比方星期六-9月-29日-2001年. 下面的例子展示了如何完成这个工作:  import java.text.SimpleDateFormat;  import java.util.Date;  public class DateExample2 {  public static void main(String[] args) {  SimpleDateFormat bartDateFormat =  new SimpleDateFormat("EEEE-MMMM-dd-yyyy");  Date date = new Date();  System.out.println(bartDateFormat.format(date));  }  }  只要通过向SimpleDateFormat 的构造函数传递格式字符串"EEE-MMMM-dd-yyyy", 我们就能够指明自己想要的格式. 你应该可以看见, 格式字符串中的ASCII 字符告诉格式化函数下面显示日期数据的哪一个部分. EEEE是星期, MMMM是月, dd是日, yyyy是年. 字符的个数决定了日期是如何格式化的.传递"EE-MM-dd-yy"会显示 Sat-09-29-01. 请察看Sun 公司的Web 站点获取日期格式化选项的完整的指示.  三、将文本数据解析成日期对象r  假设我们有一个文本字符串包含了一个格式化了的日期对象, 而我们希望解析这个字符串并从文本日期数据创建一个日期对象. 我们将再次以格式化字符串"MM-dd-yyyy" 调用SimpleDateFormat类, 但是这一次, 我们使用格式化解析而不是生成一个文本日期数据. 我们的例子, 显示在下面, 将解析文本字符串"9-29-2001"并创建一个值为001736000000 的日期对象.  例子程序:  import java.text.SimpleDateFormat;  import java.util.Date;  public class DateExample3 {  public static void main(String[] args) {  // Create a date formatter that can parse dates of  // the form MM-dd-yyyy.  SimpleDateFormat bartDateFormat =  new SimpleDateFormat("MM-dd-yyyy");  // Create a string containing a text date to be parsed.  String dateStringToParse = "9-29-2001";  try {  // Parse the text version of the date.  // We have to perform the parse method in a  // try-catch construct in case dateStringToParse  // does not contain a date in the format we are expecting.  Date date = bartDateFormat.parse(dateStringToParse);  // Now send the parsed date as a long value  // to the system output.  System.out.println(date.getTime());  }  catch (Exception ex) {  System.out.println(ex.getMessage());  }  }  }  五、使用标准的日期格式化过程  既然我们已经可以生成和解析定制的日期格式了, 让我们来看一看如何使用内建的格式化过程. 方法 DateFormat.getDateTimeInstance() 让我们得以用几种不同的方法获得标准的日期格式化过程. 在下面的例子中, 我们获取了四个内建的日期格式化过程. 它们包括一个短的, 中等的, 长的, 和完整的日期格式.  import java.text.DateFormat;  import java.util.Date;  public class DateExample4 {  public static void main(String[] args) {  Date date = new Date();  DateFormat shortDateFormat =  DateFormat.getDateTimeInstance(  DateFormat.SHORT,  DateFormat.SHORT);  DateFormat mediumDateFormat =  DateFormat.getDateTimeInstance(  DateFormat.MEDIUM,  DateFormat.MEDIUM);  DateFormat longDateFormat =  DateFormat.getDateTimeInstance(  DateFormat.LONG,  DateFormat.LONG);  DateFormat fullDateFormat =  DateFormat.getDateTimeInstance(  DateFormat.FULL,  DateFormat.FULL);  System.out.println(shortDateFormat.format(date));  System.out.println(mediumDateFormat.format(date));  System.out.println(longDateFormat.format(date));  System.out.println(fullDateFormat.format(date));  }  }  注意我们在对 getDateTimeInstance的每次调用中都传递了两个值. 第一个参数是日期风格, 而第二个参数是时间风格. 它们都是基本数据类型int(整型). 考虑到可读性, 我们使用了DateFormat 类提供的常量: SHORT, MEDIUM, LONG, 和 FULL. 要知道获取时间和日期格式化过程的更多的方法和选项, 请看Sun 公司Web 站点上的解释.  运行我们的例子程序的时候, 它将向标准输出设备输出下面的内容:  9/29/01 8:44 PM  Sep 29, 2001 8:44:45 PM  September 29, 2001 8:44:45 PM EDT  Saturday, September 29, 2001 8:44:45 PM EDT  当然，并不一定要在外部插入时间，因为在数据库中，可以让它自动插入，比如：MSSQL里面，用  getdate()来插入当前时间，而在Insert时，便可以不用管它了。但有的时候还是避免不了，要手工插入  时间，以更新数据库。  六、java.util.Calendar 格式化时间  Calendar cal = new GregorianCalendar();  int year = cal.get(Calendar.YEAR);  int month = cal.get(Calendar.MONTH)+1;  int day = cal.get(Calendar.DAY\_OF\_MONTH);  int dayOfWeek = cal.get(Calendar.DAY\_OF\_WEEK);  String week = "";  switch(dayOfWeek) {  case 1:  week = "星期天"; break;  case 2:  week = "星期一"; break;  case 3:  week = "星期二"; break;  case 4:  week = "星期三"; break;  case 5:  week = "星期四"; break;  case 6:  week = "星期五"; break;  default:  week = "星期六"; break;  int hour = cal.get(Calendar.HOUR\_OF\_DAY); // 24小时制  // int hour = cal.get(Calendar.HOUR); // 12小时制  int minute = cal.get(Calendar.MINUTE);  int second = cal.get(Calendar.SECOND);  String h,m,s;  if(hour<10) h = "0"+hour; else h = hour+"";  if(minute<10) m = "0"+minute; else m = minute+"";  if(second<10) s = "0"+second; else s = second+"";  在JSP中输出是：  今天是: 年月日 ::  结果： 今天是: 2006年4月14日星期五 05:35:26  2、在数据库中插入时间  PreparedStatement ps = con.prepareStatement("insert into TableName(dAddTime) values(?)");  这里有三种方式：  1) ps.setDate(1,new java.sql.Date(System.currentTimemillis()));  2) ps.setTime(2,new java.sql.Time(System.currentTimemillis()));  3) ps.setTimestamp(3,new java.sql.Timestamp(System.currentTimemillis()));  第一种只插入年月日 0000-00-00  常用格式的实现：  SimpleDateFormat formatterhwy=new SimpleDateFormat("yyyy-MM-dd");  java.util.Date syscurrentime=new java.util.Date();  String currentsysdate=formatterhwy.format(syscurrentime);  SimpleDateFormat myformatter=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  java.util.Date mycurrentime=new java.util.Date();  String mycurrentdate=myformatter.format(mycurrentime);  SimpleDateFormat formatterms=new SimpleDateFormat("yyyy-MM-dd HH:mm");  java.util.Date currentimems=new java.util.Date();  String currentdatems=formatterms.format(currentimems);  SimpleDateFormat formatterms1=new SimpleDateFormat("yyyyMMdd");  java.util.Date currentimems1=new java.util.Date();  String currentdatems1=formatterms1.format(currentimems1);  long today=System.currentTimeMillis();  java.util.Date todaytime=new java.util.Date(today);  SimpleDateFormat formattertoday=new SimpleDateFormat("yyyy-MM-dd");//今天的日期  String todaydate=formattertoday.format(todaytime);  long yestoday=today-86400000;  java.util.Date yestodaytime=new java.util.Date(yestoday);  SimpleDateFormat formatteryestoday=new SimpleDateFormat("yyyy-MM-dd");//昨天的日期  String yestodaydate=formatteryestoday.format(yestodaytime);  long lasthour=today-3600000;  java.util.Date lasthourtime=new java.util.Date(lasthour);  SimpleDateFormat formattershour=new SimpleDateFormat("yyyy-MM-dd HH:00:00");//某天的上一个小时开始时间  SimpleDateFormat formatterehour=new SimpleDateFormat("yyyy-MM-dd HH:59:59");//某天的上一个小时结束时间  String lastshourdate=formattershour.format(lasthourtime);  String lastehourdate=formatterehour.format(lasthourtime);  long fristday=System.currentTimeMillis(); //每月的第一天  java.util.Date fristdaytime=new java.util.Date(fristday);  SimpleDateFormat formatterfristday=new SimpleDateFormat("yyyy-MM-dd");//昨天的日期  String fristdate=formatterfristday.format(fristdaytime);  if(fristdate.substring(fristdate.length()-2).equals("01")){  //是  }  例子：  Calendar ca=Calendar.getInstance();  java.util.Date nowdate = new java.util.Date();//得到当前时间  ca.setTime(nowdate);  ca.set(Calendar.DAY\_OF\_YEAR, ca.get(Calendar.DAY\_OF\_YEAR)-1);//你想要前几天的，就减去几天  Date now = new Date(ca.getTimeInMillis());//这个就是前n天的时间  public static String GetWeekSAndE(int year,int weeki){//返回周开始日期和结束日期 用逗号隔开  String sdate="";  String edate="";  Calendar cal = Calendar.getInstance();  cal.clear();  cal.set(Calendar.YEAR, year);  cal.set(Calendar.WEEK\_OF\_YEAR, weeki);  cal.add(Calendar.DAY\_OF\_YEAR, 1);  sdate=cal.get(Calendar.YEAR)+"-"+(cal.get(Calendar.MONTH)+1)+"-"+cal.get(Calendar.DATE);  cal.clear();  cal.set(Calendar.YEAR, year);  cal.set(Calendar.WEEK\_OF\_YEAR, weeki);  cal.add(Calendar.WEEK\_OF\_YEAR, 1);  edate=cal.get(Calendar.YEAR)+"-"+(cal.get(Calendar.MONTH)+1)+"-"+cal.get(Calendar.DATE);  return sdate+","+edate;  }  import java.util.\*;  public class ShowDate {  public static void main(String[] args) {  Calendar calendar = new GregorianCalendar();  Date trialTime = new Date();  calendar.setTime(trialTime);  // print out a bunch of interesting things  System.out.println("ERA: " + calendar.get(Calendar.ERA));  System.out.println("YEAR: " + calendar.get(Calendar.YEAR));  System.out.println("MONTH: " + calendar.get(Calendar.MONTH));  System.out.println("WEEK\_OF\_YEAR: "  + calendar.get(Calendar.WEEK\_OF\_YEAR));  System.out.println("WEEK\_OF\_MONTH: "  + calendar.get(Calendar.WEEK\_OF\_MONTH));  System.out.println("DATE: " + calendar.get(Calendar.DATE));  System.out.println("DAY\_OF\_MONTH: "  + calendar.get(Calendar.DAY\_OF\_MONTH));  System.out  .println("DAY\_OF\_YEAR: " + calendar.get(Calendar.DAY\_OF\_YEAR));  System.out  .println("DAY\_OF\_WEEK: " + calendar.get(Calendar.DAY\_OF\_WEEK));  System.out.println("DAY\_OF\_WEEK\_IN\_MONTH: "  + calendar.get(Calendar.DAY\_OF\_WEEK\_IN\_MONTH));  System.out.println("AM\_PM: " + calendar.get(Calendar.AM\_PM));  System.out.println("HOUR: " + calendar.get(Calendar.HOUR));  System.out  .println("HOUR\_OF\_DAY: " + calendar.get(Calendar.HOUR\_OF\_DAY));  System.out.println("MINUTE: " + calendar.get(Calendar.MINUTE));  System.out.println("SECOND: " + calendar.get(Calendar.SECOND));  System.out  .println("MILLISECOND: " + calendar.get(Calendar.MILLISECOND));  System.out.println("ZONE\_OFFSET: "  + (calendar.get(Calendar.ZONE\_OFFSET) / (60 \* 60 \* 1000)));  System.out.println("DST\_OFFSET: "  + (calendar.get(Calendar.DST\_OFFSET) / (60 \* 60 \* 1000)));  System.out.println("Current Time, with hour reset to 3");  calendar.clear(Calendar.HOUR\_OF\_DAY); // so doesn't override  calendar.set(Calendar.HOUR, 3);  System.out.println("ERA: " + calendar.get(Calendar.ERA));  System.out.println("YEAR: " + calendar.get(Calendar.YEAR));  System.out.println("MONTH: " + calendar.get(Calendar.MONTH));  System.out.println("WEEK\_OF\_YEAR: "  + calendar.get(Calendar.WEEK\_OF\_YEAR));  System.out.println("WEEK\_OF\_MONTH: "  + calendar.get(Calendar.WEEK\_OF\_MONTH));  System.out.println("DATE: " + calendar.get(Calendar.DATE));  System.out.println("DAY\_OF\_MONTH: "  + calendar.get(Calendar.DAY\_OF\_MONTH));  System.out  .println("DAY\_OF\_YEAR: " + calendar.get(Calendar.DAY\_OF\_YEAR));  System.out  .println("DAY\_OF\_WEEK: " + calendar.get(Calendar.DAY\_OF\_WEEK));  System.out.println("DAY\_OF\_WEEK\_IN\_MONTH: "  + calendar.get(Calendar.DAY\_OF\_WEEK\_IN\_MONTH));  System.out.println("AM\_PM: " + calendar.get(Calendar.AM\_PM));  System.out.println("HOUR: " + calendar.get(Calendar.HOUR));  System.out  .println("HOUR\_OF\_DAY: " + calendar.get(Calendar.HOUR\_OF\_DAY));  System.out.println("MINUTE: " + calendar.get(Calendar.MINUTE));  System.out.println("SECOND: " + calendar.get(Calendar.SECOND));  System.out  .println("MILLISECOND: " + calendar.get(Calendar.MILLISECOND));  System.out.println("ZONE\_OFFSET: "  + (calendar.get(Calendar.ZONE\_OFFSET) / (60 \* 60 \* 1000))); // in  System.out.println("DST\_OFFSET: "  + (calendar.get(Calendar.DST\_OFFSET) / (60 \* 60 \* 1000))); // in  }  } | |