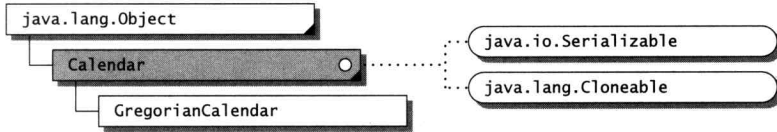


java.util
Calendar

语法

```
public abstract class Calendar implements Serializable, Cloneable
```

描述

Calendar类能够支持不同的的日历系统。它提供了多数日历系统具有的一般功能，并且使这些功能对子类可用。它的子类被用于与一个特定日历系统的日历相关的计算。例如，可以使用GregorianCalendar，它是该类的子类，来计算在格列高里历(阳历)系统中2000年12月25日是星期几。

关于该类的详细内容请见《The Java Class Libraries, Second Edition, Volume 1》。

版本1.2中所做的修改

这个类中有两个新方法：getActualMaximum()和getActualMinmum()，还增加了roll()的一个重载形式以及重载了equals()、hash Code()和toString()方法。详细信息参见这些方法的描述。

在类中包括了一种新方法，用于处理域设置不全的问题。在版本 1.1中，当希望获得足够的信息以构建一个日期(见《The Java Class Libraries, Second Edition, Volume 1》中的Calendar.set())时，所有的域将以按一种特定的顺序被检查。当对列表中位置较底的域进行修改，该方法将造成许多混乱，但可能看不到任何改变。例如，如果改变了DAY_OF_YEAR域，而后对日期重新进行计算时。为了看到变化，必须在重新计算日期前对月所在的域进行清除。

在版本1.2中一个时间戳加到了这些域上，现在，当改变一个域并且用新信息对一个日期进行修改，Calendar对象将使用最近被改变的域。下面的示例显示了这种改变的效果。

```
Calendar cal = new GregorianCalendar();
System.out.println( cal.getTime().toString() );

cal.set(Calendar.DAY_OF_YEAR, 180);
System.out.println( cal.getTime().toString() );
```

在版本1.1中该程序的结果是：

```
Sun Jan 17 15:11:08 PST 1999
Sun Jan 17 15:11:08 PST 1999
```

在版本1.2中该程序的结果是：

```
Sun Jan 17 15:11:08 PST 1999
Tue Jun 29 15:11:08 PST 1999
```






对于两个域进行设置将会导致它们与被用于日期计算的其他域之间的二义性。例如，如果对DAY_OF_WEEK域进行设置，WEEK_OF_MONTH或DAY_OF_WEEK_IN_MONTH可能被

用于计算。在这种情况下，算法将使用在 Calendar.set()中描述的顺序（见《The Java Class Libraries，Second Edition，Volume 1》）。所以，如果设置了DAY_OF_WEEK域，算法将使用WEEK_OF_MONTH域计算新的日期。

成员概述	
构造函数	
Calendar()	被子类用于创建一个Calendar实例。
创建方法	
getInstance()	创建一个Calendar实例。
月常数	
APRIL	四月。
AUGUST	八月。
DECEMBER	十二月。
FEBRUARY	二月。
JANUARY	一月。
JULY	七月。
JUNE	六月。
MARCH	三月。
MAY	五月。
NOVEMBER	十一月。
OCTOBRE	十月。
SEPTEMBER	九月。
UNDECIMBER	人造的名字代表第十三个月，用于阴历。
星期几常数	
FRIDAY	星期五。
MONDAY	星期一。
SATURDAY	星期六。
SUNDAY	星期日。
THURSDAY	星期四。
TUESDAY	星期二。
WEDNESDAY	星期三。
日历域常数	
AM_PM	A.M/P.M域。
DATE	日期
DAY_OF_MONTH	几号。
DAY_OF_WEEK	星期几。
DAY_OF_WEEK_OF_MONTH	月中星期几。
DAY_OF_YEAR	一年中的第几天。
DST_OFFSET	白天持续的时间。
ERA	纪元。
FIELD_COUNT	域的数量。

(续)

成员概述

HOUR	小时(0 ~ 12)。
HOUR_OF_DAY	小时(0 ~ 24)。
MILLISECOND	毫秒。
MINUTE	分钟。
MONTH	月。
SECOND	秒。
WEEK_OF_MONTH	月中第几个星期。
YEAR	年。
ZONE_OFFSET	与零时的偏移量。
时间常数	
AM	代表上午。
PM	代表下午。
比较方法	
 after()	判断该日历的时间是否比另一个对象中的时间晚。
 before()	判断该日历的时间是否比另一个对象中的时间早。
日历域的方法	
add()	添加/删去一个域。
clear()	清除时间域的值。
get()	获取一个域的值。
getTime()	计算日历的日期值。
isSet()	判断域是否被设置。
 roll()	滚动一个域。
set()	用于设置年、月、日、小时、分钟、秒域的值。
setTime()	设置当前的日期和时间。
日历属性方法	
getFirstDayOfWeek()	返回该Calendar对象的第一天是星期几。
getMinimalDaysInFirstWeek()	获取一年的第一个星期有几天。
isLenient()	判断该Calendar是否是lenient。
setFirstDayOfWeek()	设置星期的第一天。
setLenient()	设置该Calendar的leniency。
setMinimalDaysInFirstWeek()	设置一年的第一个星期有几天。
范围方法	
 getActualMaximum()	用当前的日期计算一个域的最大值。
 getActualMinimum()	用当前的日期计算一个域的最小值。
getGreatestMinimum()	获取一个日历域的最大的最小值。
getLeastMaximum()	获取一个日历域的最小的最大值。
getMaximum()	获取一个日历域的最大值。
getMinimum()	获取一个日历域的最小值。

(续)

成员概述	
地区方法	
getAvailableLocales()	获取一个当前日期在其中可以打印的地区集。
时区方法	
getTimeZone()	获取该Calendar的时区。
setTimezone()	设置该Calendar的时区。
受保护的方法	
complete()	计算Date或该日历域的值。
computeFields()	通过该Calendar的Date的值计算日期域的值。
computeTime()	把日期域转换成一个Date的值。
getTimeInMillis()	计算该Calendar的Date的值，并且以一个long返回它。
internalGet()	获取一个日期域的当前值。
setTimeInMillis()	设置该Calendar的当前Date的值。
受保护的域	
areFieldsSet	说明所有的域是否已经被清除。
fields	存有日历域值。
isSet	存有每一个日历域的设置状态。
isTimeSet	说明时间域是否有效。
Time	存有该Calendar当前的日期和时间。
对象方法	
clone()	创建该Calendar的一个克隆。
equals()	判断该Calendar是否与其他对象相等。
hashCode()	计算该日历的哈希码。
toString()	产生代表该Calendar的字符串。

参见

GregorianCalendar。
《The Java Class Libraries , Second Edition , Volume 1》中的Calendar。

△ after()

目的	判断该Calendar的时间是否比另一个对象中的时间晚。
语法	public boolean after (Object cal)
描述	该方法用于判断该Calendar的时间是否比cal中的时间晚。 如果 cal 是该Calendar的一个实例 并且 getTimeInMillis() > cal.getTimeInMillis()。注意：这种比较是独立于时区的。 因为需要，在进行比较之前，使用它们的日历域对 Calendar和cal的时间域进行了重新计算。
版本1.2中的改动	在版本1.1中该方法是抽象方法。
参数	
cal	一个可能为空的对象。

返回	如果该Calendar的时间比cal中的时间晚，将返回true。
异常	
IllegalArgumentException	如果有对于重新计算该Calendar和cal日历域无效的信息。
参见	before()。
示例	

```
Calendar cal1 = Calendar.getInstance();
Calendar cal2 = Calendar.getInstance(TimeZone.getTimeZone("GMT"));

// Make sure both calendars have identical times.
Date date = new Date();
cal1.setTime(date);
cal2.setTime(date);
// Demonstrates that calendar times are independent to the time zone.
System.out.println(cal1.after(cal2)); // false
System.out.println(cal2.after(cal1)); // false

cal2.set(1000, Calendar.JANUARY, 1);
System.out.println(cal1.after(cal2)); // true

cal2.set(3000, Calendar.JANUARY, 1);
System.out.println(cal1.after(cal2)); // false
```

▲ before()

目的	判断该Calendar的时间是否比另一个对象中的时间早。
语法	public boolean before (Object cal)
描述	该方法用于判断该Calendar的时间是否比cal中的时间早。 如果cal是该Calendar的一个实例并且getTimeInMillis() < cal.getTimeInMillis()。注意：这种比较是独立于时区的。 因为需要，在进行比较之前，使用它们的日历域对Calendar和cal的时间域进行了重新计算。
版本1.2中的改动	在版本1.1中该方法是抽象方法。
参数	
cal	一个可能为空的对象。
返回	如果该Calendar的时间比cal中的时间早，将返回true。
异常	
IllegalArgumentException	如果有对于重新计算该Calendar和cal日历域无效的信息。
参见	after()。
示例	

```
Calendar cal1 = Calendar.getInstance();
Calendar cal2 = Calendar.getInstance(TimeZone.getTimeZone("GMT"));

// Make sure both calendars have identical times.
Date date = new Date();
cal1.setTime(date);
cal2.setTime(date);
```

```
// Demonstrates that calendar times are independent to the time zone.
System.out.println(cal1.before(cal2));           // false
System.out.println(cal2.before(cal1));           // false

cal2.set(1000, Calendar.JANUARY, 1);
System.out.println(cal1.before(cal2));           // false
cal2.set(3000, Calendar.JANUARY, 1);
System.out.println(cal1.before(cal2));           // true
```

△ equals()

目的	判断该Calendar是否与另一个对象相等。
语法	boolean equals(Object obj)
描述	该方法用于判断该Calendar是否与对象obj相等。如果obj 是一个Calendar，它的类是该类的一个实例，并且它的属性是可识别的，该方法将返回true。用于比较的属性有：星期的第一天、第一个星期的最小天数、时区、leniency和实际日期。
版本1.2中的改动	在版本1.1中该方法是抽象方法。
参数	
obj	一个可能为空的对象。
返回	如果obj是一个Calendar，并且对于该Calendar它的属性是可识别的，那么将返回true；否则返回false。
重载	java.lang.Object.equals()。
示例	

```
Calendar cal1 = Calendar.getInstance();
Calendar cal2 = (GregorianCalendar)cal1.clone();

System.out.println(cal1.equals(null));           // false
System.out.println(cal1.equals(cal2));           // true

// Create one with a different time zone.
cal2 = Calendar.getInstance(TimeZone.getTimeZone("GMT"));
cal2.setTime(cal1.getTime());
System.out.println(cal1.equals(cal2));           // false
```

1.2 getActualMaximum()

目的	使用当前日期计算一个域的最大值。
语法	public int getActualMaximum(int fld)
描述	该方法使用当前日期计算域 fld的最大值。它与 getMaximum()有所不同，getMaximum()返回fld域的最大值时并不考虑当前日期。例如，如果现在是二月，该方法将返回 28，如果现在是一月，该方法将返回31。 该方法通过向上滚动该域(见roll())来确定返回。它的子类应该实现一个更有效的算法。
参数	
fld	一个有效的日历域常数。
返回	使用当前日期计算出域fld的最大值。
参见	getActualMinimum()、getLeastMaximum()、getGreatestMinimum()、

getMaximum()、getMinimum()。

示例

```
Calendar cal = Calendar.getInstance();
System.out.println( cal.getTime().toString() );
// Sun Jan 10 23:05:23 PST 1999

System.out.println( cal.getActualMinimum(Calendar.DAY_OF_MONTH) ); // 1
System.out.println( cal.getActualMaximum(Calendar.DAY_OF_MONTH) ); // 31

cal.set(Calendar.MONTH, Calendar.FEBRUARY);
System.out.println( cal.getActualMinimum(Calendar.DAY_OF_MONTH) ); // 1
System.out.println( cal.getActualMaximum(Calendar.DAY_OF_MONTH) ); // 28
```

1.2 getActualMinimum()

目的	使用当前日期计算一个域的最小值。
语法	public int getActualMinimum(int fld)
描述	<p>该方法使用当前日期计算域 fld 的最小值。它与 getMinimum() 有所不同，getMinimum() 返回 fld 域的最小值时并不考虑当前日期。对于阳历来说，该方法的返回通常与 getMinimum() 和 getGreatestMinimum() 相同。</p> <p>该方法通过向上滚动该域(见 roll())来确定返回。它的子类应该实现一个更有效的算法。</p>
参数	
fld	一个有效的日历域常数。
返回	使用当前日期计算出域 fld 的最小值。
参见	getActualMaximum()、getLeastMaximum()、getGreatestMinimum()、getMaximum()、getMinimum()。
示例	见 getActualMaximum()。

1.3 hashCode()

目的	计算该 Calendar 的哈希码。
语法	public int hashCode()。
描述	<p>该方法用于计算 Calendar 的哈希码。一个日历的哈希码使用星期的第一天、第一个星期的最小天数、时区、leniency 属性来计算。两个相同的 Calendar 应该有相同的哈希码。但是两个不相同的 Calendar 也可能有相同的哈希码。哈希码通常被用做哈希码表中的关键字。</p> <p>注意：哈希码的计算不包括日期。这样就可以修改一个哈希码表中的一个 Calendar 的日期。该特性的不足就是这样计算出来的哈希码不够理想，因为几乎所有的 Calendar 将被放在同一个哈希码表桶中。</p>
版本 1.2 中的改动	在版本 1.1 中该方法从 java.lang.Object 中继承。
返回	该对象的哈希码。
重载	java.lang.Object.hashCode()。
参见	equals()、getFirstDayOfWeek()、getMinimalDaysInFirstWeek()、Hashtable。

示例

```
GregorianCalendar cal = new GregorianCalendar();
System.out.println( cal.hashCode() );    // 584930835

// Hashcode is independent to date.
cal.set(1980, Calendar.JANUARY, 1);
System.out.println( cal.hashCode() );    // 584930835
```

roll()

目的

对一个日历域进行滚动。

语法

```
abstract public void roll( int fld , boolean up )
```

从版本 1.2

描述

```
public void roll( int fld , int amt)
```

该方法对日历域 fld 进行加或减。如果 up 是 true，该域就加 1。如果为 false，该域就减 1。如果 amt 为正，该域就加 amt。如果为负，就减 - amt。

如果 up 为 true，并且 get(fld) == getMaximum(fld)，该日历域就设置为 getMinimum(fld)。如果 up 为 false，并且 get(fld) == getMinimum(fld)，该日历域就设置为 getMaximum(fld)。该过程叫做滚动(rolling)。

对一个日历域的滚动也能影响到其他的日历域。例如，如果当前的日期是 1 月 31 号，并且 WEEK_OF_MONTH 域为 5，向上滚动 DATE 域将使 WEEK_OF_MONTH 域变为 1。

一个 Calendar 的 leniency 对该方法没有影响。域 fld 不能为 DST_OFFSET 或 ZONE_OFFSET。

版本 1.2 中的改动

在版本 1.2 中增加了一个带 amt 参数的重载。

参数

amt

一个正数或负数，用于对日历域进行增减。

fld

一个有效的日历域常数。

up

如果为 true，日历域向上滚动 1；如果为 false，则向下滚动 1。

异常

IllegalArgumentException

如果域 fld 为 DST_OFFSET 或 ZONE_OFFSET。

参见

add()。

示例

```
Calendar cal = Calendar.getInstance();
DateFormat df = DateFormat.getInstance();

cal.set(1997, Calendar.JANUARY, 31);

System.out.println(df.format(cal.getTime()));    // 1/31/97 9:54 AM
cal.roll(Calendar.MONTH, false);
System.out.println(df.format(cal.getTime()));    // 12/31/97 9:54 AM
cal.roll(Calendar.DATE, true);
System.out.println(df.format(cal.getTime()));    // 12/1/97 9:54 AM

cal.set(1997, Calendar.JANUARY, 31);

System.out.println(df.format(cal.getTime()));    // 1/31/97 9:54 AM
cal.roll(Calendar.MONTH, -2);
System.out.println(df.format(cal.getTime()));    // 11/30/97 9:54 AM
cal.roll(Calendar.DATE, 2);
```



```
//cal.roll(Calendar.DST_OFFSET, true);           // IllegalArgumentException
//cal.roll(Calendar.ZONE_OFFSET, true);          // IllegalArgumentException
```

▲ toString()

目的 产生一个字符串代表该Calendar。

语法 public String toString()

描述 该方法构造一个字符串代表该 Calendar的所有状态。这些状态包括：在Date值的一种形式(见Date)下的日期、时区、leniency设置、星期的第一天、第一个星期的最小天数和所有域的值。该方法常用于调试。

版本1.2中的改动

在版本1.1中，该方法从java.lang.Object中继承而来。

返回 一个代表该Calendar的非空字符串。

重载 java.lang.Object.toString()。

示例

```
Calendar cal = Calendar.getInstance();
System.out.println( cal.toString() );
```

输出

```
java.util.GregorianCalendar[time=916620401553,areFieldsSet=true,areAllFieldsSet=true,
lenient=true,zone=java.util.SimpleTimeZone[id=America/Los_Angeles,offset=-28800000,
dstSavings=3600000,useDaylight=true,startYear=0,startMode=3,startMonth=3,
startDay=1,startDayOfWeek=1,startTime=7200000,endMode=2,endMonth=9,endDay=-1,
endDayOfWeek=1,endTime=7200000],firstDayOfWeek=1,minimalDaysInFirstWeek=1,ERA=1,
YEAR=1999,MONTH=0,WEEK_OF_YEAR=4,WEEK_OF_MONTH=4,DAY_OF_MONTH=17,DAY_OF_YEAR=17,
DAY_OF_WEEK=1,DAY_OF_WEEK_IN_MONTH=3,AM_PM=1,HOURL=4,HOUR_OF_DAY=16,MINUTE=46,
SECOND=41,MILLISECOND=553,ZONE_OFFSET=-28800000,DST_OFFSET=0]
```