https://www.iteye.com/blog/josh-persistence-2230341

Java时区处理之夏令时，冬令时 - 美国的6个时区

**博客分类：**

* [Java](https://www.iteye.com/category/282305)

[Java时区](http://www.iteye.com/blogs/tag/Java%E6%97%B6%E5%8C%BA)[夏令时](http://www.iteye.com/blogs/tag/%E5%A4%8F%E4%BB%A4%E6%97%B6)[冬令时](http://www.iteye.com/blogs/tag/%E5%86%AC%E4%BB%A4%E6%97%B6)[美国6大时区](http://www.iteye.com/blogs/tag/%E7%BE%8E%E5%9B%BD6%E5%A4%A7%E6%97%B6%E5%8C%BA)

        在开始之前，如果要了解Java中的时区操作的基本概念和事例，可以参见另一篇博客：[Java时区处理之Date,Calendar,TimeZone,SimpleDateFormat](https://www.iteye.com/admin/blogs/2230074)

一、夏令时概述：

**Daylight Saving Time：DST**    夏令时，又称“日光节约时制”或“夏时制”，是一种为节约能源而人为规定地方时间的制度，在这一制度实行期间所采用的统一时间称为“夏令时间”。一般在天亮早的夏季人为将时间提前一小时，可以使人早起早睡，减少照明量，以充分利用光照资源，从而节约照明用电。各个采纳夏令时的国家具体规定不同。目前全世界有近110个国家每年要实行夏令时。（各时区多数位于其理想边界之西，导致实际上全年实施夏令时。）

      1986年至1991年，中华人民共和国在全国范围实行了六年夏令时，每年从4月中旬的第一个星期日2时整（北京时间）到9月中旬第一个星期日的凌晨2时 整（北京夏令时）。除1986年因是实行夏令时的第一年，从5月4日开始到9月14日结束外，其它年份均按规定的时段施行。1992年4月5日后不再实 行。其实，1916年，德国首先实行夏令时，英国因为怕德国会从中得到更大的效益，因此紧跟着也采取了夏令时。日本现在没有实行夏令时，但是在美国驻军的1948年到1951年使用了夏令时。另外自2011年3月27日开始俄罗斯永久使用夏令时，把时间拨快一小时，不再调回。 夏令时会导致某一天多出一个小时，或者少出一个小时。

二、实例

**1、先来看一个有趣的实例：1986年5月4号0点不见了**

**Java代码  [收藏代码](javascript:void())**

1. **package** com.wsheng.aggregator.timezone;
3. **import** java.text.DateFormat;
4. **import** java.text.SimpleDateFormat;
5. **import** java.util.Date;
7. /\*\*
8. \*
9. \* 1986年5月4号的0点不见了
10. \*
11. \* @author Josh Wang(Sheng)
12. \*
13. \* @email  swang6@ebay.com
14. \*
15. \*/
16. **public** **class** DST1 {
18. **public** **static** **final** DateFormat Y2MD\_HMS = **new** SimpleDateFormat("yy-MM-dd HH:mm:ss");
20. **public** **static** **void** main(String[] args) **throws** Exception {
21. Date d = DST1.Y2MD\_HMS.parse("1986-5-4 0:0:0");
22. System.out.println(d);
23. }
25. }

**执行结果：**

**Results代码  [收藏代码](javascript:void())**

1. Sun May 04 01:00:00 CDT 1986

**从上面的概述可以知道，这是因为中国在当天还在使用夏令时，时间被拨快了1个小时。**

**2、再来一个实例：预期的0点也没有了**

**Java代码  [收藏代码](javascript:void())**

1. **package** com.wsheng.aggregator.timezone;
3. **import** java.text.DateFormat;
4. **import** java.text.SimpleDateFormat;
5. **import** java.util.Calendar;
6. **import** java.util.Date;
8. /\*\*
9. \* @author Josh Wang(Sheng)
10. \*
11. \* @email  swang6@ebay.com
12. \*
13. \*/
14. **public** **class** DST2 {
16. **public** **static** **void** main(String[] args) **throws** Exception {
17. DateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:SS");
18. String sTime = "1991-04-07 00:00:00";
19. Date time = sdf.parse(sTime);
20. Calendar cd = Calendar.getInstance();
21. cd.setTime(time);
22. cd.add(Calendar.DATE, 7);
23. time = cd.getTime();
24. System.out.println(sdf.format(time));
25. }
27. }

**预期结果：**1991-04-14 **00:00:00**

**实际结果：**  1991-04-14 **01:00:00**

从上面的分析也很容易得出，这也是由于当时的时间中国正在实习夏令时，使用 Calendar.DST\_OFFSET 可以获得偏移量，其实夏令时是通过 TimeZone 的子类 sun.util.calendar.ZoneInfo 实现的。 ZoneInfo 中的数据存放于 %JRE\_HOME%/lib/zi 目录中。**（注意，是在jre的目录下）**我们机器一般采用的时区名称为 Asia/Shanghai，相对应的时区信息文件位于：%JRE\_HOME%/lib/zi/Asia/Shanghai 中，这是一个二进制文件。可通过 sun.util.calendar.ZoneInfoFile 的解析，可以获得需要采用夏令时的时间为：

**Results代码  [收藏代码](javascript:void())**

1. 1940-06-03 01:00:00 ~ 1940-09-30 23:00:00
2. 1941-03-16 01:00:00 ~ 1941-09-30 23:00:00
3. 1986-05-04 01:00:00 ~ 1986-09-13 23:00:00
4. 1987-04-12 01:00:00 ~ 1987-09-12 23:00:00
5. 1988-04-10 01:00:00 ~ 1988-09-10 23:00:00
6. 1989-04-16 01:00:00 ~ 1989-09-16 23:00:00
7. 1990-04-15 01:00:00 ~ 1990-09-15 23:00:00
8. 1991-04-14 01:00:00 ~ 1991-09-14 23:00:00

前段表示那一天开始的时间，也就是说那一天的 0 点在该时区中是不存在。

后段表示那一天结束的时间，由于每个时区的**夏令时**都不一样，编程语言中的**夏令时**算法比较复杂，如在北京时区：

如果当前时间为：1986-09-13 23:00:00，增加 1 小时后为 1986-09-14 00:00:00   
如果当前时间为：1986-09-13 22:00:00，增加 2 小时后为 1986-09-13 23:00:00，因为 1986-09-13 23:00:00 是**夏令时**结束时间，时钟需要回拨。

**3、更具体的例子说明：美国洛杉矶的夏令时和冬令时**

3.1、Java中不是每天都是标准的24个小时，可能是23，也可能是25。23小时和25小时就是夏令时、冬令时引起的。

3.2、下面用一个实例来判断中国和洛杉矶在1970年以来到从开始到往后的5年内对夏令时和冬令时的使用情况：

**Java代码  [收藏代码](javascript:void())**

1. **package** com.wsheng.aggregator.timezone;
3. **import** java.text.DateFormat;
4. **import** java.text.SimpleDateFormat;
5. **import** java.util.Calendar;
6. **import** java.util.Date;
7. **import** java.util.TimeZone;
9. /\*\*
10. \* 测试看看中国和美国洛杉矶在1970年以来到从现在开始的5年后的时间里，什么时候
11. \* 使用夏令时，什么时候没有使用夏令时
12. \*
13. \* @author Josh Wang(Sheng)
14. \*
15. \* @email  swang6@ebay.com
16. \*
17. \*/
18. **public** **class** DST3 {

21. **private** **static** **void** testDayTime(TimeZone timeZone) {
22. System.out.println("Time Zone is : " + timeZone.getDisplayName() + " : " + timeZone.getID());
24. DateFormat df = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
26. Calendar start = Calendar.getInstance(timeZone);
27. start.setTime(**new** Date(0)); // UTC 1970-01-01
28. System.out.println("start=" + df.format(start.getTime()));  // will print: start=1970-01-01 08:00:00
30. Calendar end = Calendar.getInstance(timeZone);
31. end.add(Calendar.YEAR, 5);
32. System.out.println("end=" + df.format(end.getTime()));

35. **boolean** find = **false**;
36. **for** (**long** i = start.getTimeInMillis(); i < end.getTimeInMillis(); i = start.getTimeInMillis()) {
37. start.add(Calendar.DATE, 1); // add one day
39. **if** ((start.getTimeInMillis() - i) % (24 \* 3600 \* 1000L) != 0) { // 是否能被24整除
40. find = **true**;
42. System.out.println("from " + df.format(**new** Date(i)) + " to " + df.format(start.getTime()) + " has "
43. + (start.getTimeInMillis() - i) + "ms" + "[" + (start.getTimeInMillis() - i) / (3600 \* 1000L)
44. + "hours]");
45. }
47. }
49. **if** (!find) {
50. System.out.println("Every day is ok.");
51. }
52. }
54. **public** **static** **void** main(String[] args) {
55. TimeZone timeZone = TimeZone.getDefault();
56. DST3.testDayTime(timeZone);
58. System.out.println(" -------------------- ");
59. timeZone = TimeZone.getTimeZone("GMT");
60. DST3.testDayTime(timeZone);
61. System.out.println(" -------------------- ");
63. timeZone = TimeZone.getTimeZone("America/Los\_Angeles");
64. DST3.testDayTime(timeZone);
65. }
67. }

输出结果：

**Results代码  [收藏代码](javascript:void())**

1. Time Zone is : China Standard Time : Asia/Shanghai
2. start=1970-01-01 08:00:00
3. end=2020-07-26 16:24:55
4. from 1986-05-03 08:00:00 to 1986-05-04 08:00:00 has 82800000ms[23hours]
5. from 1986-09-13 08:00:00 to 1986-09-14 08:00:00 has 90000000ms[25hours]
6. from 1987-04-11 08:00:00 to 1987-04-12 08:00:00 has 82800000ms[23hours]
7. from 1987-09-12 08:00:00 to 1987-09-13 08:00:00 has 90000000ms[25hours]
8. from 1988-04-09 08:00:00 to 1988-04-10 08:00:00 has 82800000ms[23hours]
9. from 1988-09-10 08:00:00 to 1988-09-11 08:00:00 has 90000000ms[25hours]
10. from 1989-04-15 08:00:00 to 1989-04-16 08:00:00 has 82800000ms[23hours]
11. from 1989-09-16 08:00:00 to 1989-09-17 08:00:00 has 90000000ms[25hours]
12. from 1990-04-14 08:00:00 to 1990-04-15 08:00:00 has 82800000ms[23hours]
13. from 1990-09-15 08:00:00 to 1990-09-16 08:00:00 has 90000000ms[25hours]
14. from 1991-04-13 08:00:00 to 1991-04-14 08:00:00 has 82800000ms[23hours]
15. from 1991-09-14 08:00:00 to 1991-09-15 08:00:00 has 90000000ms[25hours]
16. --------------------
17. Time Zone is : Greenwich Mean Time : GMT
18. start=1970-01-01 08:00:00
19. end=2020-07-26 16:24:55
20. Every day is ok.
21. --------------------
22. Time Zone is : Pacific Standard Time : America/Los\_Angeles
23. start=1970-01-01 08:00:00
24. end=2020-07-26 16:24:55
25. from 1970-04-26 08:00:00 to 1970-04-27 07:00:00 has 82800000ms[23hours]
26. from 1970-10-25 07:00:00 to 1970-10-26 08:00:00 has 90000000ms[25hours]
27. from 1971-04-25 08:00:00 to 1971-04-26 07:00:00 has 82800000ms[23hours]
28. from 1971-10-31 07:00:00 to 1971-11-01 08:00:00 has 90000000ms[25hours]
29. from 1972-04-30 08:00:00 to 1972-05-01 07:00:00 has 82800000ms[23hours]
30. from 1972-10-29 07:00:00 to 1972-10-30 08:00:00 has 90000000ms[25hours]
31. from 1973-04-29 08:00:00 to 1973-04-30 07:00:00 has 82800000ms[23hours]
32. from 1973-10-28 07:00:00 to 1973-10-29 08:00:00 has 90000000ms[25hours]
33. from 1974-01-06 08:00:00 to 1974-01-07 07:00:00 has 82800000ms[23hours]
34. from 1974-10-27 07:00:00 to 1974-10-28 08:00:00 has 90000000ms[25hours]
35. from 1975-02-23 08:00:00 to 1975-02-24 07:00:00 has 82800000ms[23hours]
36. from 1975-10-26 07:00:00 to 1975-10-27 08:00:00 has 90000000ms[25hours]
37. from 1976-04-25 08:00:00 to 1976-04-26 07:00:00 has 82800000ms[23hours]
38. from 1976-10-31 07:00:00 to 1976-11-01 08:00:00 has 90000000ms[25hours]
39. from 1977-04-24 08:00:00 to 1977-04-25 07:00:00 has 82800000ms[23hours]
40. from 1977-10-30 07:00:00 to 1977-10-31 08:00:00 has 90000000ms[25hours]
41. from 1978-04-30 08:00:00 to 1978-05-01 07:00:00 has 82800000ms[23hours]
42. from 1978-10-29 07:00:00 to 1978-10-30 08:00:00 has 90000000ms[25hours]
43. from 1979-04-29 08:00:00 to 1979-04-30 07:00:00 has 82800000ms[23hours]
44. from 1979-10-28 07:00:00 to 1979-10-29 08:00:00 has 90000000ms[25hours]
45. from 1980-04-27 08:00:00 to 1980-04-28 07:00:00 has 82800000ms[23hours]
46. from 1980-10-26 07:00:00 to 1980-10-27 08:00:00 has 90000000ms[25hours]
47. from 1981-04-26 08:00:00 to 1981-04-27 07:00:00 has 82800000ms[23hours]
48. from 1981-10-25 07:00:00 to 1981-10-26 08:00:00 has 90000000ms[25hours]
49. from 1982-04-25 08:00:00 to 1982-04-26 07:00:00 has 82800000ms[23hours]
50. from 1982-10-31 07:00:00 to 1982-11-01 08:00:00 has 90000000ms[25hours]
51. from 1983-04-24 08:00:00 to 1983-04-25 07:00:00 has 82800000ms[23hours]
52. from 1983-10-30 07:00:00 to 1983-10-31 08:00:00 has 90000000ms[25hours]
53. from 1984-04-29 08:00:00 to 1984-04-30 07:00:00 has 82800000ms[23hours]
54. from 1984-10-28 07:00:00 to 1984-10-29 08:00:00 has 90000000ms[25hours]
55. from 1985-04-28 08:00:00 to 1985-04-29 07:00:00 has 82800000ms[23hours]
56. from 1985-10-27 07:00:00 to 1985-10-28 08:00:00 has 90000000ms[25hours]
57. from 1986-04-27 08:00:00 to 1986-04-28 07:00:00 has 82800000ms[23hours]
58. from 1986-10-26 07:00:00 to 1986-10-27 08:00:00 has 90000000ms[25hours]
59. from 1987-04-05 08:00:00 to 1987-04-06 07:00:00 has 82800000ms[23hours]
60. from 1987-10-25 07:00:00 to 1987-10-26 08:00:00 has 90000000ms[25hours]
61. from 1988-04-03 08:00:00 to 1988-04-04 07:00:00 has 82800000ms[23hours]
62. from 1988-10-30 07:00:00 to 1988-10-31 08:00:00 has 90000000ms[25hours]
63. from 1989-04-02 08:00:00 to 1989-04-03 07:00:00 has 82800000ms[23hours]
64. from 1989-10-29 07:00:00 to 1989-10-30 08:00:00 has 90000000ms[25hours]
65. from 1990-04-01 08:00:00 to 1990-04-02 07:00:00 has 82800000ms[23hours]
66. from 1990-10-28 07:00:00 to 1990-10-29 08:00:00 has 90000000ms[25hours]
67. from 1991-04-07 08:00:00 to 1991-04-08 07:00:00 has 82800000ms[23hours]
68. from 1991-10-27 07:00:00 to 1991-10-28 08:00:00 has 90000000ms[25hours]
69. from 1992-04-05 08:00:00 to 1992-04-06 07:00:00 has 82800000ms[23hours]
70. from 1992-10-25 07:00:00 to 1992-10-26 08:00:00 has 90000000ms[25hours]
71. from 1993-04-04 08:00:00 to 1993-04-05 07:00:00 has 82800000ms[23hours]
72. from 1993-10-31 07:00:00 to 1993-11-01 08:00:00 has 90000000ms[25hours]
73. from 1994-04-03 08:00:00 to 1994-04-04 07:00:00 has 82800000ms[23hours]
74. from 1994-10-30 07:00:00 to 1994-10-31 08:00:00 has 90000000ms[25hours]
75. from 1995-04-02 08:00:00 to 1995-04-03 07:00:00 has 82800000ms[23hours]
76. from 1995-10-29 07:00:00 to 1995-10-30 08:00:00 has 90000000ms[25hours]
77. from 1996-04-07 08:00:00 to 1996-04-08 07:00:00 has 82800000ms[23hours]
78. from 1996-10-27 07:00:00 to 1996-10-28 08:00:00 has 90000000ms[25hours]
79. from 1997-04-06 08:00:00 to 1997-04-07 07:00:00 has 82800000ms[23hours]
80. from 1997-10-26 07:00:00 to 1997-10-27 08:00:00 has 90000000ms[25hours]
81. from 1998-04-05 08:00:00 to 1998-04-06 07:00:00 has 82800000ms[23hours]
82. from 1998-10-25 07:00:00 to 1998-10-26 08:00:00 has 90000000ms[25hours]
83. from 1999-04-04 08:00:00 to 1999-04-05 07:00:00 has 82800000ms[23hours]
84. from 1999-10-31 07:00:00 to 1999-11-01 08:00:00 has 90000000ms[25hours]
85. from 2000-04-02 08:00:00 to 2000-04-03 07:00:00 has 82800000ms[23hours]
86. from 2000-10-29 07:00:00 to 2000-10-30 08:00:00 has 90000000ms[25hours]
87. from 2001-04-01 08:00:00 to 2001-04-02 07:00:00 has 82800000ms[23hours]
88. from 2001-10-28 07:00:00 to 2001-10-29 08:00:00 has 90000000ms[25hours]
89. from 2002-04-07 08:00:00 to 2002-04-08 07:00:00 has 82800000ms[23hours]
90. from 2002-10-27 07:00:00 to 2002-10-28 08:00:00 has 90000000ms[25hours]
91. from 2003-04-06 08:00:00 to 2003-04-07 07:00:00 has 82800000ms[23hours]
92. from 2003-10-26 07:00:00 to 2003-10-27 08:00:00 has 90000000ms[25hours]
93. from 2004-04-04 08:00:00 to 2004-04-05 07:00:00 has 82800000ms[23hours]
94. from 2004-10-31 07:00:00 to 2004-11-01 08:00:00 has 90000000ms[25hours]
95. from 2005-04-03 08:00:00 to 2005-04-04 07:00:00 has 82800000ms[23hours]
96. from 2005-10-30 07:00:00 to 2005-10-31 08:00:00 has 90000000ms[25hours]
97. from 2006-04-02 08:00:00 to 2006-04-03 07:00:00 has 82800000ms[23hours]
98. from 2006-10-29 07:00:00 to 2006-10-30 08:00:00 has 90000000ms[25hours]
99. from 2007-03-11 08:00:00 to 2007-03-12 07:00:00 has 82800000ms[23hours]
100. from 2007-11-04 07:00:00 to 2007-11-05 08:00:00 has 90000000ms[25hours]
101. from 2008-03-09 08:00:00 to 2008-03-10 07:00:00 has 82800000ms[23hours]
102. from 2008-11-02 07:00:00 to 2008-11-03 08:00:00 has 90000000ms[25hours]
103. from 2009-03-08 08:00:00 to 2009-03-09 07:00:00 has 82800000ms[23hours]
104. from 2009-11-01 07:00:00 to 2009-11-02 08:00:00 has 90000000ms[25hours]
105. from 2010-03-14 08:00:00 to 2010-03-15 07:00:00 has 82800000ms[23hours]
106. from 2010-11-07 07:00:00 to 2010-11-08 08:00:00 has 90000000ms[25hours]
107. from 2011-03-13 08:00:00 to 2011-03-14 07:00:00 has 82800000ms[23hours]
108. from 2011-11-06 07:00:00 to 2011-11-07 08:00:00 has 90000000ms[25hours]
109. from 2012-03-11 08:00:00 to 2012-03-12 07:00:00 has 82800000ms[23hours]
110. from 2012-11-04 07:00:00 to 2012-11-05 08:00:00 has 90000000ms[25hours]
111. from 2013-03-10 08:00:00 to 2013-03-11 07:00:00 has 82800000ms[23hours]
112. from 2013-11-03 07:00:00 to 2013-11-04 08:00:00 has 90000000ms[25hours]
113. from 2014-03-09 08:00:00 to 2014-03-10 07:00:00 has 82800000ms[23hours]
114. from 2014-11-02 07:00:00 to 2014-11-03 08:00:00 has 90000000ms[25hours]
115. from 2015-03-08 08:00:00 to 2015-03-09 07:00:00 has 82800000ms[23hours]
116. from 2015-11-01 07:00:00 to 2015-11-02 08:00:00 has 90000000ms[25hours]
117. from 2016-03-13 08:00:00 to 2016-03-14 07:00:00 has 82800000ms[23hours]
118. from 2016-11-06 07:00:00 to 2016-11-07 08:00:00 has 90000000ms[25hours]
119. from 2017-03-12 08:00:00 to 2017-03-13 07:00:00 has 82800000ms[23hours]
120. from 2017-11-05 07:00:00 to 2017-11-06 08:00:00 has 90000000ms[25hours]
121. from 2018-03-11 08:00:00 to 2018-03-12 07:00:00 has 82800000ms[23hours]
122. from 2018-11-04 07:00:00 to 2018-11-05 08:00:00 has 90000000ms[25hours]
123. from 2019-03-10 08:00:00 to 2019-03-11 07:00:00 has 82800000ms[23hours]
124. from 2019-11-03 07:00:00 to 2019-11-04 08:00:00 has 90000000ms[25hours]
125. from 2020-03-08 08:00:00 to 2020-03-09 07:00:00 has 82800000ms[23hours]

**从输出结果可以看出：中国时区在1992年后就废除了夏令时、冬令时，GMT时区是不存在夏令时、冬令时，而美国洛杉矶一直在用夏令时、冬令时。**

**4、事例：好奇一天怎么会变23，25小时，继续试验。**

在开始该实例时，有必要交代一下美国的时区，美国横跨西五区至西十区，共六个时区。每个时区对应一个标准时间，从东向西分别为东部时间(EST)(西五区时间)、中部时间(CST)(西六区时间)、山地时间(MST)(西七区时间)、[太平洋时间](http://baike.baidu.com/view/3188740.htm" \t "_blank)(西部时间)(PST)(西八区时间)、[阿拉斯加](http://baike.baidu.com/view/4160.htm" \t "_blank)时间(AKST)(西九区时间)和夏威夷时间(HST)(西十区时间)，按照“东早西晚”的规律，各递减一小时。（由于美国使用夏令时，所以这些时区可以看做是冬令时，即夏令时结束后回拨回来的标准时间， 冬令时是在冬天使用的[标准时间](http://baike.baidu.com/view/652606.htm" \t "_blank)。在使用[日光节约时制](http://baike.baidu.com/view/587928.htm)的地区，夏天时钟拨快一[小时](http://baike.baidu.com/view/7781.htm" \t "_blank)，冬天再拨回来。这时采用的是**标准时间**，也就是冬令时。）

|  |  |
| --- | --- |
| 美国标准时间 | 与北京时间时差(小时) |
| HST | -18 |
| AKST | -17 |
| PST | -16 |
| MST | -15 |
| CST | -14 |
| EST | -13 |

如上标准时间（冬令时）一 一对应的夏令时为：

夏季始于每年4月的第1个周日，止于每年10月的最后一个周日。也称为DST: Daylight Saving Time 。。

|  |  |
| --- | --- |
| 美国夏季时间 | 与北京时间时差(小时) |
| HDT | -17 |
| AKDT | -16 |
| PDT | -15 |
| MDT | -14 |
| CDT | -13 |
| EDT | -12 |

比如对于太平洋时区而言，标准时区（冬令时时区）为PST，夏令时时区为PDT。这个地方需要注意的是CST（中部时区），CST除了代表美国的西部时区外，还可以代表别的3个时区，即CST却同时可以代表如下 4 个不同的时区：

**时区代码  [收藏代码](javascript:void())**

1. Central Standard Time (USA) UT-6:00
2. Central Standard Time (Australia) UT+9:30
3. China Standard Time UT+8:00
4. Cuba Standard Time UT-4:00

可见，CST可以同时表示美国，澳大利亚，中国，古巴四个国家的标准时间。 有时做前后端开发的时候会碰到一些奇怪的问题，比如：

      在服务器端通过 Java 获取当前时间为 Fri Aug 28 09:37:46 CST 2009， 转化为  
GMT时间为：28 Aug 2009 01:37:46 GMT，也就是说GMT时间加上 8 个小时等于CST表示的时间， 那这个  
CST不就是北京时间么，因为我们是在东八区的。 一切看起来很正常，不过在客户端用JavaScript解析这个时间就有问题了：    
    // Fri Aug 28 2009 23:37:46 GMT+0800  
   new Date('Fri Aug 28 09:37:46 CST 2009').toString();  
    好奇怪，这次GMT和CST表示的时间居然相差整整 14 个小时？

这就是因为CST可以代表多个时区引起来的，通过 Java 获取的CST时间用的是China Standard Time，而客户端JavaScript则默认采用的是美国的中部时间，所以整整相差了14个小时，  
所以将 Fri Aug 28 09:37:46 CST 2009 加上 6 个小时，再加上 8 个小时，就等于 Fri Aug 28 2009 23:37:46  
GMT+0800    
 可见，在以后的编程中为了避免错误，还是不要使用CST时间，而尽量采用GMT时间。

下面回到我们要说的例子上来：

**Java代码  [收藏代码](javascript:void())**

1. **package** com.wsheng.aggregator.timezone;
3. **import** java.text.DateFormat;
4. **import** java.text.SimpleDateFormat;
5. **import** java.util.Date;
6. **import** java.util.TimeZone;
8. /\*\*
9. \* 为什么一天会变成23、25小时
10. \*
11. \* @author Josh Wang(Sheng)
12. \*
13. \* @email  swang6@ebay.com
14. \*
15. \*/
16. **public** **class** DST4 {
18. **private** **static** DateFormat df = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
20. **public** **static** **void** main(String[] args) {
21. // 特殊时间点
22. **long** abc = 1414918799000l;
23. printDate(abc);
24. abc = 1414918800000l;
25. printDate(abc);
26. }
28. **private** **static** **void** printDate(**long** now) {
29. // 中国时间
30. // TimeZone.setDefault(TimeZone.getTimeZone("Asia/Shanghai"));
31. // System.out.println(new Date(now).toString());
32. // System.out.println(new Date(now).toLocaleString());
33. // 美国时间
34. TimeZone.setDefault(TimeZone.getTimeZone("America/Los\_Angeles"));
35. Date date = **new** Date(now);
36. System.out.println(date);
37. System.out.println(df.format(date));
38. System.out.println("============================================================");
39. }
41. }

执行结果：

**Results代码  [收藏代码](javascript:void())**

1. Sun Nov 02 01:59:59 PDT 2014
2. 2014-11-02 16:59:59
3. ============================================================
4. Sun Nov 02 01:00:00 PST 2014
5. 2014-11-02 17:00:00
6. ============================================================

发现long值为1414918799000l，美国时间是2014-11-2 1:59:59；但增加一秒1414918800000l，美国时间却是2014-11-2 1:00:00，这不是错误，仔细观察下你会发现，1414918799000l是夏令时PDT的 Sun Nov 02 01:59:59 PDT 2014；而1414918800000l是冬令时PST的 Sun Nov 02 01:00:00 PST 2014。

至此问题问题就明白了！！！

**解决方案：**

在Java中，想到Date和时区有关系，那只要在日期转换前设置下时区就解决问题了。

**Java代码  [收藏代码](javascript:void())**

1. **private** **static** String conventTime(**long** time) {
2. // 中国时间
3. // TimeZone.setDefault(TimeZone.getTimeZone("Asia/Shanghai"));
4. // 美国时间
5. TimeZone.setDefault(TimeZone.getTimeZone("America/Los\_Angeles"));
6. **return** **new** Date(time).toLocaleString();
7. }

**结论**

1. 原来Java中不是每天都是标准的24个小时，可能是23，也可能是25

2. 日期的计算，使用Calendar提供的API，是不会出差错的，简单的new Date(long milliseconds)并不靠谱

3. 来自多方协作的项目，最好使用统一的时间标准，例如系统时间，或是统一时区