7.6 Java8 的日期特性？ (2017-12-3-wl)

Java 8 日期/时间特性

Java 8 日期/时间 API 是 JSR-310 的实现，它的实现目标是克服旧的日期时间实现中所有的缺陷，新的日期/时间 API 的一些设计原则是：

- 不变性：新的日期/时间 API 中，所有的类都是不可变的，这对多线程环境有好处。
- 关注点分离：新的 API 将人可读的日期时间和机器时间（unix timestamp）明确分离，它为日期（Date）、时间（Time）、日期时间（DateTime）、时间戳（unix timestamp）以及时区定义了不同的类。
- 清晰：在所有的类中，方法都被明确定义用以完成相同的行为。举个例子，要拿到当前实例我们可以使用 now()方法，在所有的类中都定义了 format()和 parse()方法，而不是像以前那样专门有一个独立的类。为了更好的处理问题，所有的类都使用了工厂模式和策略模式，一旦你使用了其中某个类的方法，与其他类协同工作并不困难。
- 实用操作：所有新的日期/时间 API 类都实现了一系列方法用以完成通用的任务，如：加、减、格式化、解析、从日期/时间中提取单独部分，等等。
- 可扩展性：新的日期/时间 API 是工作在 ISO-8601 日历系统上的，但我们也可以将其应用在非 ISO 的日历上。

Java 8 日期/时间 API 包解释

- java.time 包：这是新的 Java 日期/时间 API 的基础包，所有的主要基础类都是这个包的一部分，如：LocalDate, LocalTime, LocalDateTime, Instant, Period, Duration 等等。所有这些类都是不可变的和线程安全的，在绝大多数情况下，这些类能够有效地处理一些公共的需求。
- java.time.chrono 包：这个包为非 ISO 的日历系统定义了一些泛化的 API，我们可以扩展 AbstractChronology 类来创建自己的日历系统。
- java.time.format 包：这个包包含能够格式化和解析日期时间对象的类，在绝大多数情况下，我们不应该直接使用它们，因为 java.time 包中相应的类已经提供了格式化和解析的方法。
- java.time.temporal 包：这个包包含一些时态对象，我们可以用其找出关于日期/时间对象的某个特定日期或时间，比如说，可以找到某月的第一天或最后一天。你可以非常容易地认出这些方法，因为它们都具有“withXXX”的格式。
- java.time.zone 包：这个包包含支持不同时区以及相关规则的类。

Java 8 日期/时间常用 API

1.java.time.LocalDate

LocalDate 是一个不可变的类，它表示默认格式(yyyy-MM-dd)的日期，我们可以使用 now()方法得到当前时间，也可以提供输入年份、月份和日期的输入参数来创建一个 LocalDate 实例。该类为 now()方法提供了重载方法，我们可以传入 ZoneId 来获得指定时区的日期。该类提供与 java.sql.Date 相同的功能，对于如何使用该类，我们来看一个简单的例子。

```
package com.journaldev.java8.time;

import java.time.LocalDate;
```

```
import java.time.Month;
import java.time.ZoneId;

/**
 * LocalDate Examples
 * @author pankaj
 *
 */
public class LocalDateExample {

    public static void main(String[] args) {

        //Current Date
        LocalDate today = LocalDate.now();
        System.out.println("Current Date="+today);

        //Creating LocalDate by providing input arguments
        LocalDate firstDay_2014 = LocalDate.of(2014, Month.JANUARY, 1);
        System.out.println("Specific Date="+firstDay_2014);

        //Try creating date by providing invalid inputs
        //LocalDate feb29_2014 = LocalDate.of(2014, Month.FEBRUARY, 29);
        //Exception in thread "main" java.time.DateTimeException:
        //Invalid date 'February 29' as '2014' is not a leap year

        //Current date in "Asia/Kolkata", you can get it from ZoneId javadoc
        LocalDate todayKolkata = LocalDate.now(ZoneId.of("Asia/Kolkata"));
        System.out.println("Current Date in IST="+todayKolkata);

        //java.time.zone.ZoneRulesException: Unknown time-zone ID: IST
        //LocalDate todayIST = LocalDate.now(ZoneId.of("IST"));

        //Getting date from the base date i.e 01/01/1970
        LocalDate dateFromBase = LocalDate.ofEpochDay(365);
        System.out.println("365th day from base date= "+dateFromBase);

        LocalDate hundredDay2014 = LocalDate.ofYearDay(2014, 100);
        System.out.println("100th day of 2014="+hundredDay2014);
    }
}
```

输出：

```
Current Date=2014-04-28
```

```
Specific Date=2014-01-01
Current Date in IST=2014-04-29
365th day from base date= 1971-01-01
100th day of 2014=2014-04-10
```

2.java.time.LocalTime

LocalTime 是一个不可变的类，它的实例代表一个符合人类可读格式的时间，默认格式是 hh:mm:ss.zzz。像 LocalDate 一样，该类也提供了时区支持，同时也可以传入小时、分钟和秒等输入参数创建实例，我们来看一个简单的程序，演示该类的使用方法。

```
package com.journaldev.java8.time;

import java.time.LocalTime;
import java.time.ZoneId;

/**
 * LocalTime Examples
 */
public class LocalTimeExample {

    public static void main(String[] args) {

        //Current Time
        LocalTime time = LocalTime.now();
        System.out.println("Current Time="+time);

        //Creating LocalTime by providing input arguments
        LocalTime specificTime = LocalTime.of(12,20,25,40);
        System.out.println("Specific Time of Day="+specificTime);

        //Try creating time by providing invalid inputs
        //LocalTime invalidTime = LocalTime.of(25,20);
        //Exception in thread "main" java.time.DateTimeException:
        //Invalid value for HourOfDay (valid values 0 - 23): 25

        //Current date in "Asia/Kolkata", you can get it from ZoneId javadoc
        LocalTime timeKolkata = LocalTime.now(ZoneId.of("Asia/Kolkata"));
        System.out.println("Current Time in IST="+timeKolkata);

        //java.time.zone.ZoneRulesException: Unknown time-zone ID: IST
        //LocalTime todayIST = LocalTime.now(ZoneId.of("IST"));

        //Getting date from the base date i.e 01/01/1970
        LocalTime specificSecondTime = LocalTime.ofSecondOfDay(10000);
        System.out.println("10000th second time= "+specificSecondTime);

    }
}
```

```
输出：
Current Time=15:51:45.240
Specific Time of Day=12:20:25.000000040
Current Time in IST=04:21:45.276
10000th second time= 02:46:40
```

3. java.time.LocalDateTime

LocalDateTime 是一个不可变的日期-时间对象，它表示一组日期-时间，默认格式是 yyyy-MM-dd-HH-mm-ss.zzz。它提供了一个工厂方法，接收 LocalDate 和 LocalTime 输入参数，创建 LocalDateTime 实例。我们来看一个简单的例子。

```
package com.journaldev.java8.time;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.Month;
import java.time.ZoneId;
import java.time.ZoneOffset;

public class LocalDateTimeExample {

    public static void main(String[] args) {

        //Current Date
        LocalDateTime today = LocalDateTime.now();
        System.out.println("Current DateTime="+today);

        //Current Date using LocalDate and LocalTime
        today = LocalDateTime.of(LocalDate.now(), LocalTime.now());
        System.out.println("Current DateTime="+today);

        //Creating LocalDateTime by providing input arguments
        LocalDateTime specificDate = LocalDateTime.of(2014, Month.JANUARY, 1, 10, 10, 30);
```

```
        System.out.println("Specific Date="+specificDate);

        //Try creating date by providing invalid inputs
        //LocalDateTime feb29_2014 = LocalDateTime.of(2014, Month.FEBRUARY, 28, 25,1,1);
        //Exception in thread "main" java.time.DateTimeException:
        //Invalid value for HourOfDay (valid values 0 - 23): 25

        //Current date in "Asia/Kolkata", you can get it from ZoneId javadoc
        LocalDateTime todayKolkata = LocalDateTime.now(ZoneId.of("Asia/Kolkata"));
        System.out.println("Current Date in IST="+todayKolkata);

        //java.time.zone.ZoneRulesException: Unknown time-zone ID: IST
        //LocalDateTime todayIST = LocalDateTime.now(ZoneId.of("IST"));

        //Getting date from the base date i.e 01/01/1970
        LocalDateTime dateFromBase = LocalDateTime.ofEpochSecond(10000, 0, ZoneOffset.UTC);
        System.out.println("10000th second time from 01/01/1970= "+dateFromBase);
    }
}
```

输出：

```
Current DateTime=2014-04-28T16:00:49.455
Current DateTime=2014-04-28T16:00:49.493
Specific Date=2014-01-01T10:10:30
Current Date in IST=2014-04-29T04:30:49.493
10000th second time from 01/01/1970= 1970-01-01T02:46:40
```

在所有这三个例子中，我们已经看到如果我们提供了无效的参数去创建日期/时间，那么系统会抛出java.time.DateTimeException，这是一种运行时异常，我们并不需要显式地捕获它。

同时我们也看到，能够通过传入 ZoneId 得到日期/时间数据，你可以从它的 Javadoc 中得到支持的 Zoneid 的列表，当运行以上类时，可以得到以上输出。

4. java.time.Instant

含义：从1970-01-01 00：00：00到当前时间的毫秒值

Instant 类是用在机器可读的时间格式上的，它以 Unix 时间戳的形式存储日期时间，我们来看一个简单的程序

```
package com.journaldev.java8.time;

import java.time.Duration;
import java.time.Instant;
```

```
public class InstantExample {

    public static void main(String[] args) {
        //Current timestamp
        Instant timestamp = Instant.now();
        System.out.println("Current Timestamp = "+timestamp);

        //Instant from timestamp
        Instant specificTime = Instant.ofEpochMilli(timestamp.toEpochMilli());
        System.out.println("Specific Time = "+specificTime);

        //Duration example
        Duration thirtyDay = Duration.ofDays(30);
        System.out.println(thirtyDay);
    }
}
```

输出：

```
Current Timestamp = 2014-04-28T23:20:08.489Z
Specific Time = 2014-04-28T23:20:08.489Z
PT720H
```

带时区

```
//自定义设置偏移时间
OffsetDateTime time = ins.atOffset(ZoneOffset.ofHours(8));
System.out.println(time);

//默认时区的偏移时间
OffsetDateTime now = OffsetDateTime.now();
System.out.println(now);

//获取系统默认时区时间
ZonedDateTime systemDefaultzoneDateTime = ins.atZone(ZoneId.systemDefault());
System.out.println(systemDefaultzoneDateTime);

//获取特定时区时间
ZonedDateTime zoneDateTime = ins.atZone(ZoneId.of("Asia/Kolkata"));
System.out.println(zoneDateTime);

//获取从1970-01-01  00：00：00到当前时间的秒值，区别？
System.out.println(ins.getEpochSecond());
System.out.println(ins.toEpochMilli());
```

5. 日期 API 工具

我们早些时候提到过，大多数日期/时间 API 类都实现了一系列工具方法，如：加/减天数、周数、月份数，等等。还有其他的工具方法能够使用 TemporalAdjuster 调整日期，并计算两个日期间的周期。

```
package com.journaldev.java8.time;

import java.time.LocalDate;
import java.time.LocalTime;
import java.time.Period;
import java.time.temporal.TemporalAdjusters;

public class DateAPIUtilities {

    public static void main(String[] args) {

        LocalDate today = LocalDate.now();

        //Get the Year, check if it's leap year
        System.out.println("Year "+today.getYear()+" is Leap Year? "+today.isLeapYear());
```

```
        //Compare two LocalDate for before and after
        System.out.println("Today is before 01/01/2015? "+today.isBefore(LocalDate.of(2015,1,1)));

        //Create LocalDateTime from LocalDate
        System.out.println("Current Time="+today.atTime(LocalTime.now()));

        //plus and minus operations
        System.out.println("10 days after today will be "+today.plusDays(10));
        System.out.println("3 weeks after today will be "+today.plusWeeks(3));
        System.out.println("20 months after today will be "+today.plusMonths(20));

        System.out.println("10 days before today will be "+today.minusDays(10));
        System.out.println("3 weeks before today will be "+today.minusWeeks(3));
        System.out.println("20 months before today will be "+today.minusMonths(20));

        //Temporal adjusters for adjusting the dates
         System.out.println("First date of this month= "+today.
         with(TemporalAdjusters.firstDayOfMonth()));
        LocalDate lastDayOfYear = today.with(TemporalAdjusters.lastDayOfYear());
        System.out.println("Last date of this year= "+lastDayOfYear);

        Period period = today.until(lastDayOfYear);
        System.out.println("Period Format= "+period);
        System.out.println("Months remaining in the year= "+period.getMonths());
    }
}
```

输出：

```
Year 2014 is Leap Year? false
Today is before 01/01/2015? true
Current Time=2014-04-28T16:23:53.154
10 days after today will be 2014-05-08
3 weeks after today will be 2014-05-19
20 months after today will be 2015-12-28
10 days before today will be 2014-04-18
3 weeks before today will be 2014-04-07
20 months before today will be 2012-08-28
First date of this month= 2014-04-01
Last date of this year= 2014-12-31
Period Format= P8M3D
Months remaining in the year= 8
```

6. 解析和格式化

将一个日期格式转换为不同的格式，之后再解析一个字符串，得到日期时间对象，这些都是很常见的。我们来看一下简单的例子。

```
package com.journaldev.java8.time;
import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class DateParseFormatExample {

    public static void main(String[] args) {

        //Format examples
        LocalDate date = LocalDate.now();
        //default format
        System.out.println("Default format of LocalDate="+date);
        //specific format
        System.out.println(date.format(DateTimeFormatter.ofPattern("d::MMM::uuuu")));
        System.out.println(date.format(DateTimeFormatter.BASIC_ISO_DATE));

        LocalDateTime dateTime = LocalDateTime.now();
        //default format
        System.out.println("Default format of LocalDateTime="+dateTime);
        //specific format
    System.out.println(dateTime.format(DateTimeFormatter.ofPattern("d::MMM::uuuu HH::mm::ss")));
        System.out.println(dateTime.format(DateTimeFormatter.BASIC_ISO_DATE));

        Instant timestamp = Instant.now();
        //default format
        System.out.println("Default format of Instant="+timestamp);

        //Parse examples
        LocalDateTime dt = LocalDateTime.parse("27::Apr::2014 21::39::48",
                DateTimeFormatter.ofPattern("d::MMM::uuuu HH::mm::ss"));
        System.out.println("Default format after parsing = "+dt);
    }
}
输出：
Default format of LocalDate=2014-04-28
```

这里不行，只有八月这种形式可以

```
28::Apr::2014
20140428
Default format of LocalDateTime=2014-04-28T16:25:49.341
28::Apr::2014 16::25::49
20140428
Default format of Instant=2014-04-28T23:25:49.342Z
Default format after parsing = 2014-04-27T21:39:48
```

7. 旧的日期时间支持

旧的日期/时间类已经在几乎所有的应用程序中使用，因此做到向下兼容是必须的。这也是为什么会有若干工具方法帮助我们将旧的类转换为新的类，反之亦然。我们来看一下简单的例子。

```
package com.journaldev.java8.time;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.TimeZone;

public class DateAPILegacySupport {

    public static void main(String[] args) {

        //Date to Instant
        Instant timestamp = new Date().toInstant();
        //Now we can convert Instant to LocalDateTime or other similar classes
        LocalDateTime date = LocalDateTime.ofInstant(timestamp,
                        ZoneId.of(ZoneId.SHORT_IDS.get("PST")));
        System.out.println("Date = "+date);

        //Calendar to Instant
        Instant time = Calendar.getInstance().toInstant();
        System.out.println(time);
        //TimeZone to ZoneId
        ZoneId defaultZone = TimeZone.getDefault().toZoneId();
```

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

            //ZonedDateTime from specific Calendar
            ZonedDateTime gregorianCalendarDateTime = new GregorianCalendar().toZonedDateTime();
            System.out.println(gregorianCalendarDateTime);

            //Date API to Legacy classes
            Date dt = Date.from(Instant.now());
            System.out.println(dt);

            TimeZone tz = TimeZone.getTimeZone(defaultZone);
            System.out.println(tz);

            GregorianCalendar gc = GregorianCalendar.from(gregorianCalendarDateTime);
            System.out.println(gc);

        }
    }
```

输出：

```
Date = 2014-04-28T16:28:54.340
2014-04-28T23:28:54.395Z
America/Los_Angeles
2014-04-28T16:28:54.404-07:00[America/Los_Angeles]
Mon Apr 28 16:28:54 PDT 2014
sun.util.calendar.ZoneInfo[id="America/Los_Angeles",offset=-
28800000,dstSavings=3600000,useDaylight=true,transitions=185,lastRule=java.util.SimpleTimeZone[id=A
merica/Los_Angeles,offset=-
28800000,dstSavings=3600000,useDaylight=true,startYear=0,startMode=3,startMonth=2,startDay=8,startD
ayOfWeek=1,startTime=7200000,startTimeMode=0,endMode=3,endMonth=10,endDay=1,endDayOfWeek=1,endTime=
7200000,endTimeMode=0]]
java.util.GregorianCalendar[time=1398727734404,areFieldsSet=true,areAllFieldsSet=true,lenient=t
rue,zone=sun.util.calendar.ZoneInfo[id="America/Los_Angeles",offset=-
28800000,dstSavings=3600000,useDaylight=true,transitions=185,lastRule=java.util.SimpleTimeZone[id=A
merica/Los_Angeles,offset=-
28800000,dstSavings=3600000,useDaylight=true,startYear=0,startMode=3,startMonth=2,startDay=8,startD
ayOfWeek=1,startTime=7200000,startTimeMode=0,endMode=3,endMonth=10,endDay=1,endDayOfWeek=1,endTime=
7200000,endTimeMode=0]],firstDayOfWeek=2,minimalDaysInFirstWeek=4,ERA=1,YEAR=2014,MONTH=3,WEEK_OF_Y
EAR=18,WEEK_OF_MONTH=5,DAY_OF_MONTH=28,DAY_OF_YEAR=118,DAY_OF_WEEK=2,DAY_OF_WEEK_IN_MONTH=4,AM_PM=1
,HOUR=4,HOUR_OF_DAY=16,MINUTE=28,SECOND=54,MILLISECOND=404,ZONE_OFFSET=-28800000,DST_OFFSET=3600000]
```

补充：我们可以看到，旧的 TimeZone 和 GregorianCalendar 类的 toString()方法太啰嗦了，一点都不友好。

7.7 Java8 之前的日期和时间使用的槽点（2017-11-19-wl)

Tiago Fernandez 做过一次投票，选举最烂的 JAVA API，排第一的 EJB2.X，第二的就是日期 API（**Date** 和 **Calender**）

1. 槽点一

最开始的时候，Date 既要承载日期信息，又要做日期之间的转换，还要做不同日期格式的显示，职责较繁杂（不懂单一职责，你妈妈知道吗？纯属恶搞~哈哈）

后来从 JDK 1.1 开始，这三项职责分开了：

1）使用 Calendar 类实现日期和时间字段之间转换；

2）使用 DateFormat 类来格式化和分析日期字符串；

3）而 Date 只用来承载日期和时间信息。

原有 Date 中的相应方法已废弃。不过，无论是 Date，还是 Calendar，都用着太不方便了，这是 API 没有设计好的地方。

2. 槽点二

坑爹的 **year 和 month**。

我们看下面的代码：

```
1. Date date = new Date(2012,1,1);
2. System.out.println(date);
```

输出 Thu Feb 01 00:00:00 CST 3912

观察输出结果，year 是 2012+1900，而 month，月份参数我不是给了 1 吗？怎么输出二月（Feb）了？

应该曾有人告诉你，如果你要设置日期，应该使用 java.util.Calendar，像这样...

```
1. Calendar calendar = Calendar.getInstance();
```

```
2. calendar.set(2013, 8, 2);
```

这样写又不对了，calendar 的 month 也是从 0 开始的，表达 8 月份应该用 7 这个数字，要么就干脆用枚举

```
1. calendar.set(2013, Calendar.AUGUST, 2);
```

注意上面的代码，Calendar 年份的传值不需要减去 1900（当然月份的定义和 Date 还是一样），这种不一致真是让人抓狂！有些人可能知道，**Calendar 相关的 API 是 IBM 捐出去的**，所以才导致不一致。

3. 槽点三

java.util.Date 与 java.util.Calendar 中的所有属性都是可变的

下面的代码，计算两个日期之间的天数....

```
1. public static void main(String[] args) {
2.    Calendar birth = Calendar.getInstance();
3.    birth.set(1975, Calendar.MAY, 26);
4.    Calendar now = Calendar.getInstance();
5.    System.out.println(daysBetween(birth, now));
6.    System.out.println(daysBetween(birth, now)); // 显示 0?
7. }
8.
9. public static long daysBetween(Calendar begin, Calendar end) {
10.    long daysBetween = 0;
11.   while(begin.before(end)) {
12.      begin.add(Calendar.DAY_OF_MONTH, 1);
13.      daysBetween++;
14.   }
15.   return daysBetween;
16. }
```

daysBetween 有点问题，如果连续计算两个 Date 实例的话，第二次会取得 0，因为 Calendar 状态是可变的，考虑到重复计算的场合，最好复制一个新的 Calendar

```
1. public static long daysBetween(Calendar begin, Calendar end) {
2.    Calendar calendar = (Calendar) begin.clone(); // 复制
3.    long daysBetween = 0;
4.    while(calendar.before(end)) {
5.       calendar.add(Calendar.DAY_OF_MONTH, 1);
6.       daysBetween++;
7.    }
8.    return daysBetween;
```

```
9. }
```

以上种种，导致目前有些第三方的 java 日期库诞生，比如广泛使用的 JODA-TIME，还有 Date4j 等，虽然第三方库已经足 3 / 8 够强大，好用，但还是有兼容问题的，比如标准的 JSF 日期转换器与 joda-time API 就不兼容，你需要编写自己的转换器，所以标准的 API 还是必须的，**于是就有了 JSR310。**

7.8 Java8 日期实现 JSR310 规范（2017-11-23-wl)

1. JSR310 介绍

JSR 310 实际上有两个日期概念。第一个是 Instant，它大致对应于 java.util.Date 类，因为它代表了一个确定的时间点，即相对于标准 Java 纪元（1970 年 1 月 1 日）的偏移量；但与 java.util.Date 类不同的是其精确到了纳秒级别。

第二个对应于人类自身的观念，比如 LocalDate 和 LocalTime。他们代表了一般的时区概念，要么是日期（不包含时间），要么是时间（不包含日期），类似于 java.sql 的表示方式。此外，还有一个 MonthDay，它可以存储某人的生日（不包含年份）。每个类都在内部存储正确的数据而不是像 java.util.Date 那样利用午夜 12 点来区分日期，利用 1970-01-01 来表示时间。

目前 Java8 已经实现了 JSR310 的全部内容。新增了 java.time 包定义的类表示了日期-时间概念的规则，包括 instants,durations, dates, times, time-zones and periods。这些都是基于 ISO 日历系统，它又是遵循 Gregorian 规则的。最重要的一点是值不可变，且线程安全，通过下面一张图，我们快速看下 java.time 包下的一些主要的类的值的格式，方便理解。

```
• LocalDate       2010-12-03
• LocalTime                 11:05:30
• LocalDateTime   2010-12-03T11:05:30
• OffsetTime                11:05:30+01:00
• OffsetDateTime  2010-12-03T11:05:30+01:00
• ZonedDateTime   2010-12-03T11:05:30+01:00 Europe/Paris
• Year            2010
• YearMonth       2010-12
• MonthDay            -12-03
• Instant         2576458258.266 seconds after 1970-01-01
```

2. Java8 方法概览

java.time 包下的方法概览

方法名	说明
Of	静态工厂方法
parse	静态工厂方法，关注于解析
get	获取某些东西的值
is	检查某些东西的是否是 true
with	不可变的 setter 等价物
plus	加一些量到某个对象
minus	从某个对象减去一些量
to	转换到另一个类型
at	把这个对象与另一个对象组合起来

与旧的 API 相比

Java.time ISO Calendar	Java.util Calendar
Instant	Date
LocalDate, LocalTime, LocalDateTime	Calendar
ZonedDateTime	Calendar
OffsetDateTime, OffsetTime,	Calendar
ZoneId, ZoneOffset, ZoneRules	TimeZone
Week Starts on Monday (1 .. 7) enum MONDAY, TUESDAY, … SUNDAY	Week Starts on Sunday (1 .. 7) int values SUNDAY, MONDAY, … SATURDAY
12 Months (1 .. 12) enum JANUARY, FEBRUARY, …, DECEMBER	12 Months (0 .. 11) int values JANUARY, FEBRUARY, … DECEMBER

3. 简单实用 java.time 的 API 实用

```
1. public class TimeIntroduction {
2.   public static void testClock() throws InterruptedException {
3.     //时钟提供给我们用于访问某个特定 时区的 瞬时时间、日期 和 时间的。
4.     Clock c1 = Clock.systemUTC(); //系统默认 UTC 时钟（当前瞬时时间 System.currentTimeMillis()）
5.     System.out.println(c1.millis()); //每次调用将返回当前瞬时时间（UTC）
6.     Clock c2 = Clock.systemDefaultZone(); //系统默认时区时钟（当前瞬时时间）
7.     Clock c31 = Clock.system(ZoneId.of("Europe/Paris")); //巴黎时区
8.     System.out.println(c31.millis()); //每次调用将返回当前瞬时时间（UTC）
9.     Clock c32 = Clock.system(ZoneId.of("Asia/Shanghai"));//上海时区
10.    System.out.println(c32.millis());//每次调用将返回当前瞬时时间（UTC）
11.    Clock c4 = Clock.fixed(Instant.now(), ZoneId.of("Asia/Shanghai"));//固定上海时区时钟
12.    System.out.println(c4.millis());
13.    Thread.sleep(1000);
14.
```

```
15.    System.out.println(c4.millis()); //不变 即时钟时钟在那一个点不动
16.    Clock c5 = Clock.offset(c1, Duration.ofSeconds(2)); //相对于系统默认时钟两秒的时钟
17.    System.out.println(c1.millis());
18.    System.out.println(c5.millis());
19. }
20. public static void testInstant() {
21.      //瞬时时间 相当于以前的 System.currentTimeMillis()
22.      Instant instant1 = Instant.now();
23.      System.out.println(instant1.getEpochSecond());//精确到秒 得到相对于 1970-01-01 00:00:00
24. UTC 的一个时间
25.      System.out.println(instant1.toEpochMilli()); //精确到毫秒
26.      Clock clock1 = Clock.systemUTC(); //获取系统 UTC 默认时钟
27.      Instant instant2 = Instant.now(clock1);//得到时钟的瞬时时间
28.      System.out.println(instant2.toEpochMilli());
29.      Clock clock2 = Clock.fixed(instant1, ZoneId.systemDefault()); //固定瞬时时间时钟
30.      Instant instant3 = Instant.now(clock2);//得到时钟的瞬时时间
31.      System.out.println(instant3.toEpochMilli());//equals instant1
32.  }
33.   public static void testLocalDateTime() {
34.   //使用默认时区时钟瞬时时间创建 Clock.systemDefaultZone() -->即相对于 ZoneId.systemDefault()
35.默认时区
36.   LocalDateTime now = LocalDateTime.now();
37.   System.out.println(now);
38.   //自定义时区
39.   LocalDateTime now2 = LocalDateTime.now(ZoneId.of("Europe/Paris"));
40.   System.out.println(now2);//会以相应的时区显示日期
41.    //自定义时钟
42.   Clock clock = Clock.system(ZoneId.of("Asia/Dhaka"));
43.   LocalDateTime now3 = LocalDateTime.now(clock);
44.   System.out.println(now3);//会以相应的时区显示日期
45.   //不需要写什么相对时间 如 java.util.Date 年是相对于 1900 月是从 0 开始
46.   //2013-12-31 23:59
47.   LocalDateTime d1 = LocalDateTime.of(2013, 12, 31, 23, 59);
48.   //年月日 时分秒 纳秒
49.   LocalDateTime d2 = LocalDateTime.of(2013, 12, 31, 23, 59, 59, 11);
50.   //使用瞬时时间 + 时区
51.   Instant instant = Instant.now();
52.   LocalDateTime d3 = LocalDateTime.ofInstant(Instant.now(), ZoneId.systemDefault());
53.   System.out.println(d3);
54.   //解析 String--->LocalDateTime
55.   LocalDateTime d4 = LocalDateTime.parse("2013-12-31T23:59");
56.   System.out.println(d4);
57.   LocalDateTime d5 = LocalDateTime.parse("2013-12-31T23:59:59.999");//999 毫秒 等价于
```

```
58. 999000000 纳秒
59.
60.   System.out.println(d5);
61.   //使用 DateTimeFormatter API 解析 和 格式化
62.   DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
63.   LocalDateTime d6 = LocalDateTime.parse("2013/12/31 23:59:59", formatter);
64.   System.out.println(formatter.format(d6));
65.   //时间获取
66.   System.out.println(d6.getYear());
67.   System.out.println(d6.getMonth());
68.   System.out.println(d6.getDayOfYear());
69.   System.out.println(d6.getDayOfMonth());
70.   System.out.println(d6.getDayOfWeek());
71.   System.out.println(d6.getHour());
72.   System.out.println(d6.getMinute());
73.   System.out.println(d6.getSecond());
74.   System.out.println(d6.getNano());
75.   //时间增减
76.   LocalDateTime d7 = d6.minusDays(1);
77.   LocalDateTime d8 = d7.plus(1, IsoFields.QUARTER_YEARS);
78.   //LocalDate 即年月日 无时分秒
79.   //LocalTime 即时分秒 无年月日
80.   //API 和 LocalDateTime 类似就不演示了
81.   }
82.   public static void testZonedDateTime() {
83.   //即带有时区的 date-time 存储纳秒、时区和时差（避免与本地 date-time 歧义）。
84.   //API 和 LocalDateTime 类似，只是多了时差(如 2013-12-20T10:35:50.711+08:00[Asia/Shanghai])
85.   ZonedDateTime now = ZonedDateTime.now();
86.   System.out.println(now);
87.   ZonedDateTime now2 = ZonedDateTime.now(ZoneId.of("Europe/Paris"));
88.   System.out.println(now2);
89.   //其他的用法也是类似的 就不介绍了
90.   ZonedDateTime z1 = ZonedDateTime.parse("2013-12-31T23:59:59Z[Europe/Paris]");
91.   System.out.println(z1);
92. }
93.   public static void testDuration() {
94.   //表示两个瞬时时间的时间段
95.   Duration d1 = Duration.between(Instant.ofEpochMilli(System.currentTimeMillis() - 12323123),
96.  Instant.now())
97. ;
98.   //得到相应的时差
99.   System.out.println(d1.toDays());
100.    System.out.println(d1.toHours());
```

```
101.    System.out.println(d1.toMinutes());
102.
103.    System.out.println(d1.toMillis());
104.   System.out.println(d1.toNanos());
105.   //1 天时差 类似的还有如 ofHours()
106.   Duration d2 = Duration.ofDays(1);
107.   System.out.println(d2.toDays());
108. }
109.   public static void testChronology() {
110.    //提供对 java.util.Calendar 的替换，提供对年历系统的支持
111.     Chronology c = HijrahChronology.INSTANCE;
112.     ChronoLocalDateTime d = c.localDateTime(LocalDateTime.now());
113.     System.out.println(d);
114.   }
115. /**
116. * 新旧日期转换
117. */
118.  public static void testNewOldDateConversion(){
119.   Instant instant=new Date().toInstant();
120.   Date date=Date.from(instant);
121.   System.out.println(instant);
122.   System.out.println(date);
123.  }
124.  public static void main(String[] args) throws InterruptedException {
125.   testClock();
126.   testInstant();
127.   testLocalDateTime();
128.   testZonedDateTime();
129.   testDuration();
130.   testChronology();
131.   testNewOldDateConversion();
132.  }
133. }
```

7.9 JSR310 规范 Joda-Time 的区别（2017-11-23-wl)

其实 JSR310 的规范领导者 Stephen Colebourne，同时也是 Joda-Time 的创建者，JSR310 是在 Joda-Time 的基础上建立的，参考了绝大部分的 API，但并不是说 JSR310=JODA-Time，下面几个比较明显的区别是：

1. 最明显的变化就是包名（从 org.joda.time 以及 java.time）

2. JSR310 不接受 NULL 值，Joda-Time 视 NULL 值为 0

3. JSR310 的计算机相关的时间（Instant）和与人类相关的时间（DateTime）之间的差别变得更明显

4. JSR310 所有抛出的异常都是 DateTimeException 的子类。虽然 DateTimeException 是一个 RuntimeException

7.10 总结 (2017-11-23-wl)

Java.time	java.util.Calendar 以及 Date
流畅的 API	不流畅的 API
实例不可变	实例可变
线程安全	非线程安全