1.LocalDate、 LocalTime 、 LocalDateTime

1.Java 8 吸收了 Joda Time 的精华，以一个新的开始为 Java 创建优秀的 API 。新的 java.time 中包含了所有关于 本地日期（ LocalDate ）、本地时间(LocalTime）、本地日期时间(LocalDateTime ）、时区 ZonedDateTime 和持续时间（ Duration ）的类

2.新时间日期API

java.time----包含值对象的基础包

java.time.chrono----提供对不同的日历系统的访问

java.time.format----格式化和解析时间和日期

java.time.temporal----包括底层框架和扩展特性

java.time.zone----包含时区支持的类

说明：大多数开发者只会用到基础包和format 包，也可能会用到 temporal 包。因此，尽管有 68 个新的公开类型，大多数开发者，大概将只会用到其中的三分之一。

3.LocalDate 、 LocalTime 、 LocalDateTime 类是其中较重要的几个类，它们的实例是不可变的对象 ，分别表示使用 ISO-8601 日历系统的日期、时间、日期和时间。

注：ISO-8601 日历系统是国际标准化组织制定的现代公民的日期和时间的表示法，也就是公历。

- LocalDate 代表 IOS 格式（ yyyy MM dd ）的日期 可以存储 生日、纪念日等日期。
- LocalTime 表示一个时间，而不是 日期 。
- LocalDateTime 是用来表示日期和时间的， 这是一个最常用的类之

方法	描述
now() / * now(Zoneld zone)	静态方法，根据当前时间创建对象/指定时区的对象
of()	静态方法，根据指定日期/时间创建对象
getDayOfMonth()/getDayOfYear()	获得月份天数(1-31) /获得年份天数(1-366)
getDayOfWeek()	获得星期几(返回一个 DayOfWeek 枚举值)
getMonth()	获得月份, 返回一个 Month 枚举值
getMonthValue() / getYear()	获得月份(1-12) /获得年份
getHour()/getMinute()/getSecond()	获得当前对象对应的小时、分钟、秒
withDayOfMonth()/withDayOfYear()/ withMonth()/withYear()	将月份天数、年份天数、月份、年份修改为指定的值并返回新的对象
plusDays(), plusWeeks(), plusMonths(), plusYears(),plusHours()	向当前对象添加几天、几周、几个月、几年、几小时
minusMonths() / minusWeeks()/ minusDays()/minusYears()/minusHours()	从当前对象减去几月、几周、几天、几年、几小时

```java
@Test
public void test(){
 LocalDate localdate = LocalDate.now();
 LocalTime localtime = LocalTime.now();
 LocalDateTime localDateTime = LocalDateTime.now();

 System.out.println(localdate); // 2021-08-16
 System.out.println(localtime); // 22:57:31.348
 System.out.println(localDateTime); // 2021-08-16T22:57:31.348
  System.out.println("**********************************");

 // of() :设置指定的年月日时分秒.没有偏移量
 LocalDateTime ldt = LocalDateTime.of(2021, 8, 16, 19, 57, 56);
 System.out.println(ldt); // 2021-08-16T19:57:56
 System.out.println("**********************************");

 /*
 * getXxx()
 */
 // 获得月份天数 (1~31)
 System.out.println(ldt.getDayOfMonth()); // 16
 System.out.println("**********************************");
```

```java

// 获得年份天数(1~366)
System.out.println(ldt.getDayOfYear()); // 228
System.out.println("**********************************");

// 获得星期几,返回一个 DayOfWeek 枚举值
System.out.println(ldt.getDayOfWeek()); // MONDAY
System.out.println("**********************************");

// 获得月份,返回一个 Month 枚举值
System.out.println(ldt.getMonth()); // AUGUST
System.out.println("**********************************");

// 获得月份(1~12)
System.out.println(ldt.getMonthValue()); // 8
System.out.println("**********************************");

// 获得年份
System.out.println(ldt.getYear()); // 2021
System.out.println("**********************************");


// 获得当前对象对应的小时、 分钟 、 秒
System.out.println(ldt.getHour()); // 19
System.out.println(ldt.getMinute()); // 57
System.out.println(ldt.getSecond()); // 56
System.out.println("**********************************");

/*
* 体现不可变性
* withXxx():设置相关的属性
* 将月份天数、年份天数、月份 、年份、小时修改为指定的值并返回新的对象
*/
LocalDateTime ldt1 = ldt.withDayOfMonth(18);
System.out.println(ldt); // 2021-08-16T19:57:56
System.out.println(ldt1); // 2021-08-18T19:57:56
System.out.println("**********************************");

```

```java
LocalDateTime ldt2 = ldt.withHour(22);
System.out.println(ldt); // 2021-08-16T19:57:56
System.out.println(ldt2); // 2021-08-16T22:57:56
System.out.println("**********************************");

/*
* 向当前对象添加几天 、 几周 、 几个月 、 几年 、 几小时
*/

LocalDateTime plusDays = ldt.plusDays(6);
System.out.println(ldt); // 2021-08-16T19:57:56
System.out.println(plusDays); // 2021-08-22T19:57:56
System.out.println("**********************************");

LocalDateTime weeks = ldt.plusWeeks(2);
System.out.println(ldt); // 2021-08-16T19:57:56
System.out.println(weeks); // 2021-08-30T19:57:56
System.out.println("**********************************");

LocalDateTime months = ldt.plusMonths(2);
System.out.println(ldt); // 2021-08-16T19:57:56
System.out.println(months); // 2021-10-16T19:57:56
System.out.println("**********************************");

LocalDateTime years = ldt.plusYears(2);
System.out.println(ldt); // 2021-08-16T19:57:56
System.out.println(years); // 2023-08-16T19:57:56
System.out.println("**********************************");

LocalDateTime hours = ldt.plusHours(3);
System.out.println(ldt); // 2021-08-16T19:57:56
System.out.println(hours); // 2021-08-16T22:57:56
System.out.println("**********************************");

/*
* 从当前对象减去几月 、 几周 、 几 天 、 几年 、 几小时
* */

```

```
99  LocalDateTime months1 = ldt.minusMonths(2);
100  System.out.println(ldt); // 2021-08-16T19:57:56
101  System.out.println(months1); // 2021-06-16T22:57:56
102  System.out.println("********************************");
103 }
```

2. Instant

1. Instant ：时间线上的一个瞬时点。 这可能被用来记录应用程序中的事件时间戳 。

2. 在 UNIX 中，这个数从1970年开始，以秒为的单位；同样的，在Java中，也是从1970年开始，但以毫秒为单位。时间线中的一个点表示为一个很大的数，这有利于计算机处理。

3. java.time 包通过值类型 Instant 提供机器视图，不提供处理人类意义上的时间单位。 Instant 表示时间线上的一点，而不需要任何上下文信息。概念上讲， 它只是简单的表示自 1970 年 1 月 1 日 0 时 0 分 0 秒（ UTC ）开始的秒数。 因为 java.time 包是基于纳秒计算的，所以 Instant 的精度可以达到纳秒级。

4. 1 ns = 10^(- 9) s　　1 秒 = 1000 毫秒 =10^6 微秒 =10^9 纳秒

5. 常用方法:

方法	描述
now()	静态方法，返回默认UTC时区的Instant类的对象
ofEpochMilli(long epochMilli)	静态方法，返回在1970-01-01 00:00:00基础上加上指定毫秒数之后的Instant类的对象
atOffset(ZoneOffset offset)	结合即时的偏移来创建一个 OffsetDateTime
toEpochMilli()	返回1970-01-01 00:00:00到当前时间的毫秒数，即为时间戳

时间戳是指格林威治时间1970年01月01日00时00分00秒(北京时间1970年01月01日08时00分00秒)起至现在的总秒数。

```
1 /*
2  Instant的使用
3  类似于 java.util.Date类
4 */
5 @Test
6 public void instatnt(){
7  // 静态方法,返回默认UTC时区的Instant类的对象
```

```
8   Instant instant = Instant.now();
9   System.out.println(instant); // 2021-08-16T15:17:45.765Z
10   System.out.println("*********************************");
11
12   // 添加时间的偏移量
13   OffsetDateTime offsetDateTime =
 instant.atOffset(ZoneOffset.ofHours(8));
14   System.out.println(instant); // 2021-08-16T15:17:45.765Z
15   System.out.println(offsetDateTime); // 2021-08-16T23:21:28.060+08:00
16   System.out.println("*********************************");
17
18   // toEpochMilli():获取自1970年1月1日0时0分0秒（UTC）开始的毫秒数 ---> D
 ate类的getTime()
19   long milli = instant.toEpochMilli();
20   System.out.println(milli); // 1629127397864
21   System.out.println("*********************************");
22
23   // ofEpochMilli():通过给定的毫秒数，获取Instant实例 -->Date(long milli
 s)
24   Instant ofEpochMilli = Instant.ofEpochMilli(1629127397864L);
25   System.out.println(ofEpochMilli); // 2021-08-16T15:23:17.864Z
26 }
```

3. java.time.format.DateTimeFormatter

java.time.format.DateTimeFormatter类：该类提供了三种格式化方法：

1. 预定义的标准格式。

ISO_LOCAL_DATE_TIME;ISO_LOCAL_DATE;ISO_LOCAL_TIME;

2. 本地化相关的格式。如： ofLocalizedDateTime(FormatStyle.LONG)

FormatStyle.LONG / FormatStyle.MEDIUM / FormatStyle.SHORT :适用于LocalDateTim

3. 自定义的格式。如： ofPattern(“yyyy MM dd hh:mm:ss”)

方 法	描 述
ofPattern(String pattern)	静态方法，返回一个指定字符串格式的DateTimeFormatter
format(TemporalAccessor t)	格式化一个日期、时间，返回字符串
parse(CharSequence text)	将指定格式的字符序列解析为一个日期、时间

```java
/*
* DateTimeFormatter:格式化或解析日期、时间
* 类似于SimpleDateFormat
* */
@Test
public void dateTimeFormatter(){
 // 方式一：预定义的标准格式。如：ISO_LOCAL_DATE_TIME;ISO_LOCAL_DATE;ISO_LOCAL_TIME
 DateTimeFormatter formatter = DateTimeFormatter.ISO_LOCAL_DATE_TIME;

 // 格式化 ： 日期 -- > 字符串
 LocalDateTime localDateTime = LocalDateTime.now();
 String str1 = formatter.format(localDateTime);
 System.out.println(localDateTime); // 2021-08-16T23:36:46.366
 System.out.println(str1); // 2021-08-16T23:36:46.366
 System.out.println("**********************************");

 // 解析 ： 字符串 -- > 日期
 TemporalAccessor parse = formatter.parse("2021-08-16T23:36:46.366");
 System.out.println(parse); // {},ISO resolved to 2021-08-16T23:36:46.366
 System.out.println("**********************************");

 /*
 * 方式二：
 * 本地化相关的格式。如：ofLocalizedDateTime()
 * FormatStyle.LONG / FormatStyle.MEDIUM / FormatStyle.SHORT :适用于LocalDateTime
 * */
 DateTimeFormatter formatter1 = DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG);

 // 格式化：
 String str2 = formatter1.format(localDateTime);
 System.out.println(str2); // 2021年8月17日 下午08时49分35秒
 System.out.println("**********************************");

 // 本地化相关的格式。如：ofLocalizedDate()
```

```
35  // FormatStyle.FULL / FormatStyle.LONG / FormatStyle.MEDIUM / Format
Style.SHORT : 适用于LocalDate
36  DateTimeFormatter formatter2 =
DateTimeFormatter.ofLocalizedDateTime(FormatStyle.MEDIUM);
37
38  // 格式化
39  String str3 = formatter2.format(localDateTime);
40  System.out.println(str3); // 2021-8-17 20:51:25
41  System.out.println("*********************************");
42
43  // FormatStyle.FULL / FormatStyle.LONG / FormatStyle.MEDIUM / Format
Style.SHORT : 适用于LocalDate
44  DateTimeFormatter formatter3 =
DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT);
45
46  // 格式化
47  String str4 = formatter3.format(localDateTime);
48  System.out.println(str4); // 21-8-17 下午8:52
49  System.out.println("*********************************");
50
51  /*
52  * (重点)方式三：自定义的格式。如：ofPattern(“yyyy-MM-dd hh:mm:ss”)
53  * */
54
55  DateTimeFormatter formatter4 = DateTimeFormatter.ofPattern("yyyy-MM-
dd hh:mm:ss");
56  // 格式化: 日期 -- > 字符串
57  String str5 = formatter4.format(LocalDateTime.now());
58  System.out.println(str5); // 2021-08-16 11:53:00
59
60  //解析:字符串 -- > 日期
61  TemporalAccessor accessor = formatter4.parse("2019-02-18 03:52:09");
62  System.out.println(accessor); // {MinuteOfHour=52, MilliOfSecond=0,
NanoOfSecond=0, SecondOfMinute=9, HourOfAmPm=3, MicroOfSecond=0},ISO re
olved to 2019-02-18
63
64 }
```

4. 其它类

- **ZoneId**：该类中包含了所有的时区信息，一个时区的ID，如 Europe/Paris
- **ZonedDateTime**：一个在ISO-8601日历系统时区的日期时间，如 2007-12-03T10:15:30+01:00 Europe/Paris。
  - 其中每个时区都对应着ID，地区ID都为“{区域}/{城市}”的格式，例如：Asia/Shanghai等
- **Clock**：使用时区提供对当前即时、日期和时间的访问的时钟。
- 持续时间：**Duration**，用于计算两个“时间”间隔
- 日期间隔：**Period**，用于计算两个“日期”间隔
- **TemporalAdjuster**：时间校正器。有时我们可能需要获取例如：将日期调整到“下一个工作日”等操作。
- **TemporalAdjusters**：该类通过静态方法(firstDayOfXxx()/lastDayOfXxx()/nextXxx())提供了大量的常用 TemporalAdjuster 的实现。

```
@Test
public void test1(){
 //ZoneId: 类中包含了所有的时区信息
 // ZoneId的 getAvailableZoneIds(): 获取所有的 ZoneId
 /*Set<String> zoneIds = ZoneId.getAvailableZoneIds();
 for (String s : zoneIds ){
 System.out.println(s);
 }*/
 System.out.println("**********************************");

 // ZoneId 的 of():获取指定时区的时间
 LocalDateTime localDateTime = LocalDateTime.now(ZoneId. of("Asia/Tok
yo"));
 System.out.println(localDateTime);
 System.out.println("**********************************");

 // ZonedDateTime: 带时区的日期时间
 // ZonedDateTime的 now(): 获取本时区的 ZonedDateTime 对象
 ZonedDateTime zonedDateTime = ZonedDateTime.now();
 System.out.println(zonedDateTime);
 // ZonedDateTime的 now(ZoneId id): 获取指定时区的 ZonedDateTime 对象
 ZonedDateTime zonedDateTime1 = ZonedDateTime.now(ZoneId. of("Asia/To
kyo"));
 System.out.println(zonedDateTime1);
 System.out.println("**********************************");

```

```java
    // Duration: 用于计算两个“时间”间隔，以秒和纳秒为基准
    LocalTime localTime = LocalTime.now();
    LocalTime localTime1 = LocalTime.of(15, 23,32);
    //between():静态方法，返回 Duration 对象，表示两个时间的间隔
    Duration duration = Duration.between(localTime1, localTime);
    System.out.println(duration);

    System.out.println(duration.getSeconds());
    System.out.println(duration.getNano());
    System.out.println("*******************************");

    LocalDateTime localDateTime1 = LocalDateTime.of(2016, 6, 12, 15,
23,32);
    LocalDateTime localDateTime2 = LocalDateTime.of(2017, 6, 12, 15,
23,32);
    Duration duration1 = Duration.between(localDateTime2,
localDateTime1);
    System.out.println(duration1);
}

@Test
public void test2(){
    // Period: 用于计算两个“日期”间隔，以年、月、日衡量
    LocalDate localDate = LocalDate.now();
    LocalDate localDate1 = LocalDate.of(2024,6,20);

    Period period = Period.between(localDate, localDate1);
    System.out.println(period); // P2Y10M3D
    System.out.println("*******************************");

    System.out.println(localDate);
    System.out.println(period.getYears()); // 2
    System.out.println(period.getMonths()); // 10
    System.out.println(period.getDays()); // 3
    System.out.println("*******************************");

    Period period1 = period.withYears(4);
    System.out.println(period1); // P4Y10M3D
```

```
60
61 }
```