

16赞

赏

赞赏

下载App

LocalDateTime用法

逆水寻洲

关注

1

2019.06.09 19:05:04 字数 1,618 阅读 35,515

转白：[java8 — 新日期时间API篇](#)

前言

最近看别人项目源码，发现Java8新的日期时间API很方便强大，所以转载该入门介绍博客，记录一下。

使用新时间日期API的必要性

在java8以前，或许：

- 当你在做有关时间日期的操作时，你会想到用Date;
- 当你在做日期、月份、天数相加减时，你会想到用Calendar;
- 当你需要对时间日期进行格式化时，你会想到使用SimpleDateFormat或DateFormat下的其他子类;
-

但是，你必须知道，以上有关的时间日期操作对象，都是可变的、线程不安全的，同时，如果作为一个经常写过类似代码的人来说，尽管有相关对象提供某些操作，但并不能很快、很简单的就能得到最终想要的结果，如：要计算两个时间点之间相差的年、月、日、周、时、分、秒等，这些计算尽管原有API也能够实现，但原有API除了线程不安全之外，另外一个不足之处就是代码繁琐，性能低！

为何我们总提多线程下，线程不安全？对于初学者来说，可能觉得能够简单实现出功能就已经足够，但是真正的开发项目是不可能仅仅考虑功能的实现的，还要考虑项目的安全性、稳定性、高性能、高可用性等等！因此，作为java开发者，多线程的知识是必不可少的。而也正因为多线程，才会出现一大堆问题（简称线程安全性问题），作为开发者，就应该写出不仅能实现功能的代码，还要是线程安全的代码。那么，学习并熟练掌握新的线程安全的API就显得非常重要了！

没错，java8出的新的时间日期API都是线程安全的，并且性能更好，代码更简洁！

新时间日期API常用、重要对象介绍

- ZonedId: 时区ID，用来确定Instant和LocalDateTime互相转换的规则
- Instant: 用来表示时间线上的一个点（瞬时）
- LocalDate: 表示没有时区的日期, LocalDate是不可变并且线程安全的
- LocalTime: 表示没有时区的时间, LocalTime是不可变并且线程安全的
- LocalDateTime: 表示没有时区的日期时间, LocalDateTime是不可变并且线程安全的
- Clock: 用于访问当前时刻、日期、时间，用到时区
- Duration: 用秒和纳秒表示时间的数量（长短），用于计算两个日期的“时间”间隔
- Period: 用于计算两个“日期”间隔

其中，LocalDate、LocalTime、LocalDateTime是新API里的基础对象，绝大多数操作都是围绕这几个对象来进行的，有必要搞清楚：

LocalDate：只含年月日的日期对象
LocalTime：只含时分秒的时间对象
LocalDateTime：同时含有年月日时分秒的日期对象

本文将以实例讲解日常开发中常用到的时间日期操作，如：

获取当前日期、时间
指定时间日期创建对应的对象
计算两个时间点的间隔
判断两个时间的前后
时间日期的格式化
获取时间戳
时间、日期相加减
获取给定时间点的年份、月份、周、星期等
.....

新时间日期API详解与示例

获取当前时间

```
1 | LocalDate localDate = LocalDate.now();
2 | LocalTime localTime = LocalTime.now();
3 | LocalDateTime localDateTime = LocalDateTime.now();
4 | System.out.println(localDate);
5 | System.out.println(localTime);
6 | System.out.println(localDateTime);
```

运行结果：



根据指定日期/时间创建对象

```
1 | LocalDate localDate = LocalDate.of(2018, 1, 13);
2 | LocalTime localTime = LocalTime.of(9, 43, 20);
3 | LocalDateTime localDateTime = LocalDateTime.of(2018, 1, 13, 9, 43, 20);
4 | System.out.println(localDate);
5 | System.out.println(localTime);
6 | System.out.println(localDateTime);
```

运行结果：



日期时间的加减

- 对于LocalDate,只有精度大于或等于日的加减，如年、月、日；
- 对于LocalTime,只有精度小于或等于时的加减，如时、分、秒、纳秒；
- 对于LocalDateTime,则可以进行任意精度的时间相加减；

```
1 | LocalDateTime localDateTime = LocalDateTime.now();
2 | //以下方法的参数都是long型，返回值都是LocalDateTime
3 | LocalDateTime plusYearsResult = localDateTime.plusYears(2L);
4 | LocalDateTime plusMonthsResult = localDateTime.plusMonths(3L);
```

逆水寻洲

关注

总资产11 (约0.72元)

go代码无法调试的问题
阅读 499

Golang Sync包
阅读 188

推荐阅读

- Java常用集合-Map简单介绍 (HashMap、HashTable和TreeMap)
阅读 582
- sentinel集成与sentinel配置持久化到nacos
阅读 499
- Kotlin-基本语法
阅读 433
3. 静态资源配置
阅读 134
- Spring boot代码生成器配合velocity模板
阅读 219



16 赞



赏



下载App

```
20 //System.out.println("当前时间为：" + LocalDateTime.now());
21 + "当前时间加2年后为：" + plusYearsResult + "\n"
22 + "当前时间加3个月为：" + plusMonthsResult + "\n"
23 + "当前时间加7日后为：" + plusDaysResult + "\n"
24 + "当前时间加2小时后为：" + plusHoursResult + "\n"
25 + "当前时间加10分钟后为：" + plusMinutesResult + "\n"
26 + "当前时间加10秒后为：" + plusSecondsResult + "\n"
27 );
28
29 //也可以以另一种方式来相加减日期，即plus(long amountToAdd, TemporalUnit unit)
30 // 参数1： 相加的数量， 参数2： 相加的单位
31 LocalDateTime nextMonth = LocalDateTime.plus(1, ChronoUnit.MONTHS);
32 LocalDateTime nextYear = LocalDateTime.plus(1, ChronoUnit.YEARS);
33 LocalDateTime nextWeek = LocalDateTime.plus(1, ChronoUnit.WEEKS);
34
35 System.out.println("now：" + LocalDateTime.now()
36 + "nextYear：" + nextYear + "\n"
37 + "nextMonth：" + nextMonth + "\n"
38 + "nextWeek：" + nextWeek + "\n"
39 );
40
41 //日期的减法用法一样，在此不再举例
```

运行结果：

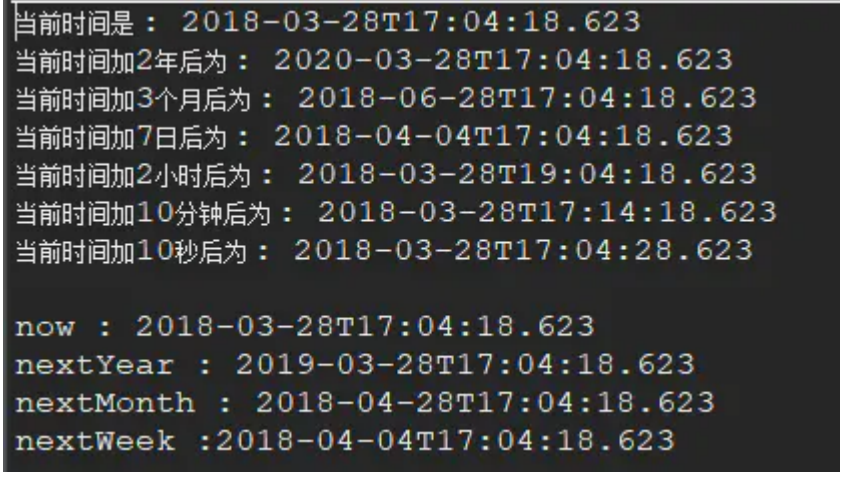


image.png

将年、月、日等修改为指定的值，并返回新的日期（时间）对象

析： 其效果与时间日期相加减差不多，如今天是2018-01-13，要想变为2018-01-20有两种方式

- a. `localDate.plusDays(20L)` -> 相加指定的天数
- b. `localDate.withDayOfYear(20)` -> 直接指定到哪一天

```
1 | LocalDateTime localDate = LocalDateTime.now();
2 | //当前时间基础上，指定本年当中的第几天，取值范围为1-365,366
3 | LocalDateTime withDayOfYearResult = localDate.withDayOfYear(200);
4 | //当前时间基础上，指定本月当中的第几天，取值范围为1-29,30,31
5 | LocalDateTime withDayOfMonthResult = localDate.withDayOfMonth(5);
6 | //当前时间基础上，直接指定年份
7 | LocalDateTime withYearResult = localDate.withYear(2017);
8 | //当前时间基础上，直接指定月份
9 | LocalDateTime withMonthResult = localDate.withMonth(5);
10 | System.out.println("当前时间是：" + localDate + "\n"
11 | + "指定本年当中的第200天：" + withDayOfYearResult + "\n"
12 | + "指定本月当中的第5天：" + withDayOfMonthResult + "\n"
13 | + "直接指定年份为2017：" + withYearResult + "\n"
14 | + "直接指定月份为5月：" + withMonthResult + "\n"
15 | );
```

运行结果：

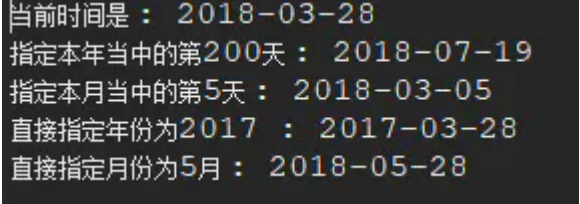


image.png

获取日期的年月日周时分秒

```
1 | LocalDateTime localDateTime = LocalDateTime.now();
2 | int dayOfYear = localDateTime.getDayOfYear();
3 | int dayOfMonth = localDateTime.getDayOfMonth();
4 | DayOfWeek dayOfWeek = localDateTime.getDayOfWeek();
5 | System.out.println("今天是" + localDateTime + "\n"
6 | + "本年当中第" + dayOfYear + "天" + "\n"
7 | + "本月当中第" + dayOfMonth + "天" + "\n"
8 | + "本周中星期" + dayOfWeek.getValue() + "-即" + dayOfWeek + "\n");
9 |
10 | //获取当天时间的年月日时分秒
11 | int year = localDateTime.getYear();
12 | Month month = localDateTime.getMonth();
13 | int day = localDateTime.getDayOfMonth();
14 | int hour = localDateTime.getHour();
15 | int minute = localDateTime.getMinute();
16 | int second = localDateTime.getSecond();
17 | System.out.println("今天是" + localDateTime + "\n"
18 | + "年：" + year + "\n"
19 | + "月：" + month.getValue() + "-即" + month + "\n"
20 | + "日：" + day + "\n"
21 | + "时：" + hour + "\n"
22 | + "分：" + minute + "\n"
23 | + "秒：" + second + "\n"
24 | );
```

运行结果：

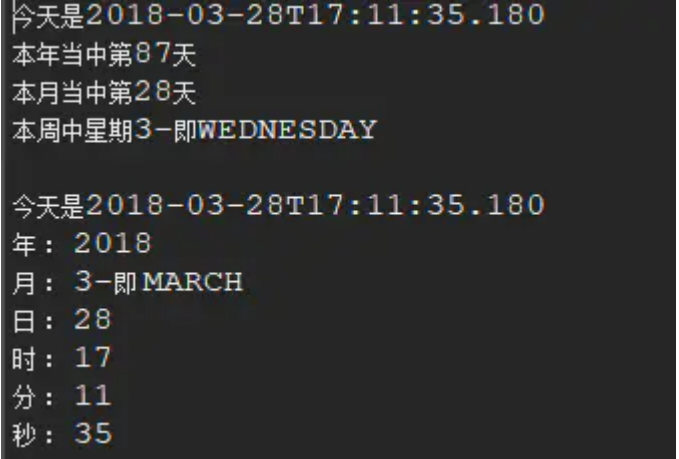


image.png

时间日期前后的比较与判断

```
1 | //判断两个时间点的前后
2 | LocalDateTime localDate1 = LocalDateTime.of(2017, 8, 8);
3 | LocalDateTime localDate2 = LocalDateTime.of(2018, 8, 8);
4 | boolean date1IsBeforeDate2 = localDate1.isBefore(localDate2);
5 | System.out.println("date1IsBeforeDate2：" + date1IsBeforeDate2);
6 | // date1IsBeforeDate2 == true
```

判断是否为闰年

```
1 | LocalDateTime now = LocalDateTime.now();
2 | System.out.println("now：" + now + "，is leap year？" + );
```

java8时钟：clock()

```
1 | //返回当前时间，根据系统时间和UTC
2 | Clock clock = Clock.systemUTC();
3 | // 运行结果： SystemClock[Z]
4 | System.out.println(clock);
```

写下你的评论...

 评论0  赞16 ...

比如Date.from(Instant)就是用来把Instant转换成java.util.date的, 而new Date().toInstant()就是将Date转换成Instant的

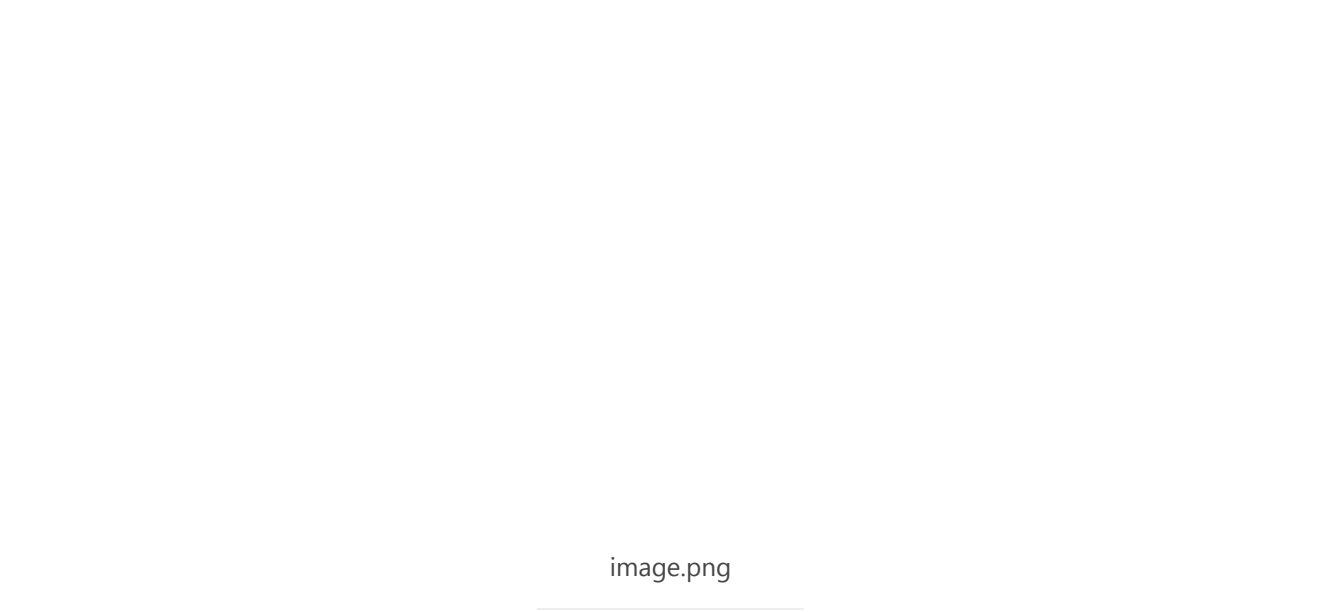
```
1 Instant instant = Instant.now();
2 //2019-06-08T16:50:19.174Z
3 System.out.println(instant);
4 Date date = Date.from(instant);
5 Instant instant2 = date.toInstant();
6 //Sun Jun 09 00:50:19 CST 2019
7 System.out.println(date);
8 //2019-06-08T16:50:19.174Z
9 System.out.println(instant2);
```

计算时间、日期间隔

Duration:用于计算两个“时间”间隔
Period:用于计算两个“日期”间隔

```
1 //计算两个日期的日期间隔-年月日
2 LocalDate date1 = LocalDate.of(2018, 2, 13);
3 LocalDate date2 = LocalDate.of(2017, 3, 12);
4 //内部是用date2-date1, 所以得到的结果是负数
5 Period period = Period.between(date1, date2);
6 System.out.println("相差年数 : " + period.getYears());
7 System.out.println("相差月数 : " + period.getMonths());
8 System.out.println("相差日数 : " + period.getDays());
9 //还可以这样获取相差的年月日
10 System.out.println("-----");
11 long years = period.get(ChronoUnit.YEARS);
12 long months = period.get(ChronoUnit.MONTHS);
13 long days = period.get(ChronoUnit.DAYS);
14 System.out.println("相差的年月日分别为 : " + years + "," + months + "," + days);
15 //注意, 当获取两个日期的间隔时, 并不是单纯的年月日对应的数字相加减, 而是会先算出具体差多少天, 在折算成相
16
17 //计算两个时间的间隔
18 System.out.println("-----");
19 LocalDateTime date3 = LocalDateTime.now();
20 LocalDateTime date4 = LocalDateTime.of(2018, 1, 13, 22, 30, 10);
21 Duration duration = Duration.between(date3, date4);
22 System.out.println(date3 + " 与 " + date4 + " 间隔 " + "\n"
23     + " 天 : " + duration.toDays() + "\n"
24     + " 时 : " + duration.toHours() + "\n"
25     + " 分 : " + duration.toMinutes() + "\n"
26     + " 毫秒 : " + duration.toMillis() + "\n"
27     + " 纳秒 : " + duration.toNanos() + "\n"
28 );
29 //注意, 并没有获得秒差的, 但既然可以获得毫秒, 秒就可以自行获取了
```

运行结果:



当计算程序的运行时间时, 应当使用时间戳Instant

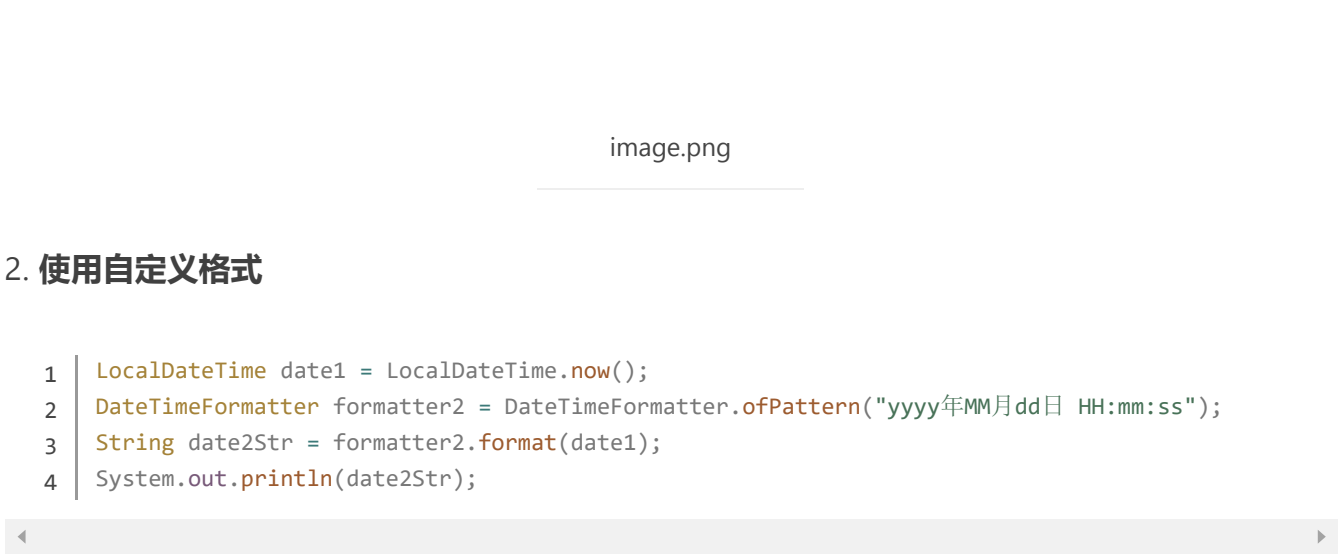
```
1 Instant ins1 = Instant.now();
2 for (int i = 0; i < 10000000; i++) {
3     //循环一百万次
4 }
5 Instant ins2 = Instant.now();
6 Duration duration = Duration.between(ins1, ins2);
7 System.out.println("程序运行耗时为 : " + duration.toMillis() + "毫秒");
```

时间日期的格式化 (格式化后返回的类型是String)

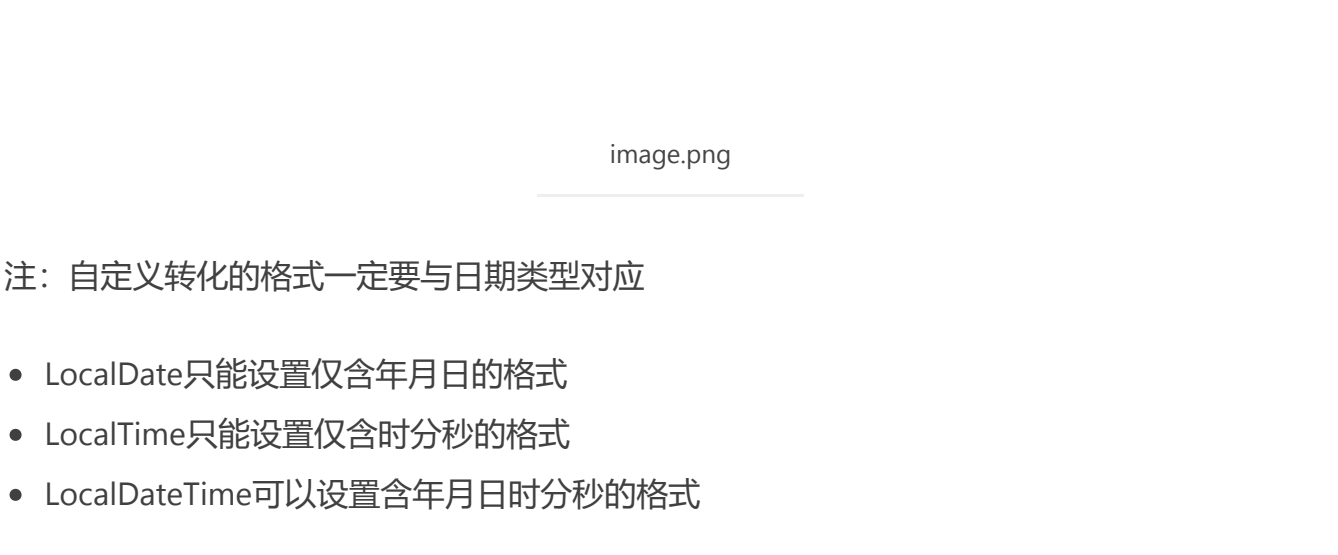
1. 使用jdk自身配置好的日期格式

```
1 //使用jdk自身配置好的日期格式
2 DateTimeFormatter formatter1 = DateTimeFormatter.ISO_DATE_TIME;
3 LocalDateTime date1 = LocalDateTime.now();
4 //反过来调用也可以 : date1.format(formatter1);
5 String date1Str = formatter1.format(date1);
6 System.out.println(date1Str);
```

运行结果:



运行结果:



代码如下:

```
1 DateTimeFormatter formatter3 = DateTimeFormatter.ofPattern("yyyy-MM-dd");
2 System.out.println(formatter3.format(LocalDate.now()));
3
4 System.out.println("-----");
5
6 DateTimeFormatter formatter4 = DateTimeFormatter.ofPattern("HH:mm:ss");
7 System.out.println(formatter4.format(LocalTime.now()));
```

运行结果:



```
1 String datetime = "2018-01-13 21:27:30";
2 DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
3 LocalDateTime ldt = LocalDateTime.parse(datetime, dtf);
4 System.out.println(ldt);
```

运行结果:



16赞



赞赏



下载App



注：格式的写法必须与字符串的形式一样

2018-01-13 21:27:30 对应 yyyy-MM-dd HH:mm:ss

20180113213328 对应 yyyyMMddHHmmss

否则会报运行时异常！

但要记住：得到的最终结果都是类似2018-01-13T21:27:30的格式

因为在输出LocalDateTime对象时，会调用其重写的toString方法。

```
1 @Override
2 public String toString() {
3     return date.toString() + 'T' + time.toString();
4 }
```

将时间日期对象转为格式化后的时间日期对象

```
1 //新的格式化API中，格式化后的结果都默认是String，有时我们也需要返回经过格式化的同类型对象
2 LocalDateTime ldt1 = LocalDateTime.now();
3 DateTimeFormatter dtf1 = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
4 String temp = dtf1.format(ldt1);
5 LocalDateTime formattedDateTime = LocalDateTime.parse(temp, dtf1);
6 System.out.println(formatedDateTime);
```

运行结果:



long毫秒值转换为日期

```
1 System.out.println("-----long毫秒值转换为日期-----");
2 DateTimeFormatter df= DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
3 String longToDateTime = df.format(LocalDateTime.ofInstant(
4     Instant.ofEpochMilli(System.currentTimeMillis()),ZoneId.of("Asia/Shanghai")));
5 System.out.println(longToDateTime);
```

运行结果:



 16人点赞 >



 java



"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下

 逆水寻洲

总资产11 (约0.72元) 共写了3.1W字 获得86个赞 共26个粉丝

 关注

-  程序员外包平台
-  庭院小鱼池
-  孩子智力低表现
-  茶颜悦色加盟
-  古茗加盟
-  数据库mysql


被以下专题收入，发现更多相似内容

 java基础知识

推荐阅读

更多精彩内容 >

跟上 Java 8：日期和时间实用技巧
当你开始使用Java操作日期和时间的时候，会有一些棘手。你也许会通过System.currentTimeMillis...

 余平的余 余平的平 阅读 1,971 评论 0 赞 2




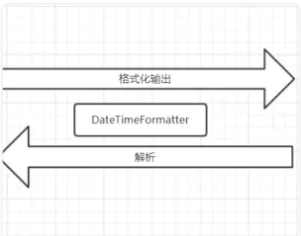
跟上 Java 8：日期和时间实用技巧
当你开始使用Java操作日期和时间的时候，会有一些棘手。你也许会通过System.currentTimeMillis...


 程序猿的那些事 阅读 1,544 评论 0 赞 1



jdk1.5-1.8新特性
原链接：http://www.cnblogs.com/langtianya/p/3757993.html JDK各...
 把爱放下会走更远 阅读 596 评论 0 赞 10

Java8：java.time api【原创】
java8 新的时间api 本篇文章分为三个部分：基础的日期时间对象的使用 操作和解析日期时间对象 基于时区的调...
 杨比轩 阅读 369 评论 0 赞 4



桥
几度西桥 我总以为我会在这里遇到你 桥下车水马龙 眼睛看花了 哪一点光亮是你 桥上秋风拂面 头发数乱了 哪一丝微凉...
 龚晓燕 阅读 84 评论 0 赞 0

