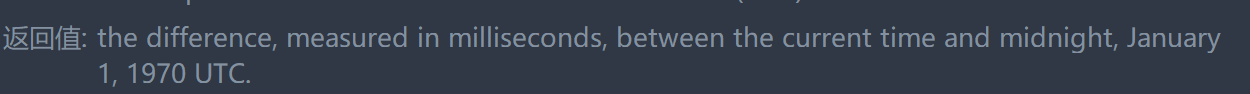
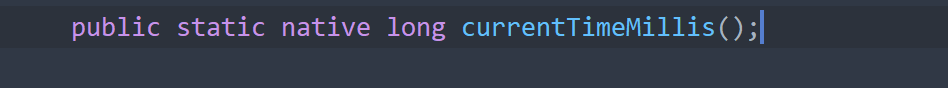
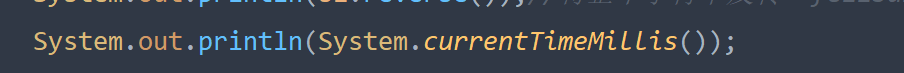
# 时间戳

System类下面有一个静态方法，返回值是long型的





意为获取1970年1月1号到当前的毫秒数。并返回一个long型的数



可以在某个方法开始调用一次用long型变量接收

在方法结束时调用一次用long型变量接收

两者相减就是这个方法运行所需要的时间。

# Date时间类

## Util包下的时间类

Date（） date=new Date（）；

空参构造器，重载的toString（）方法输出new对象的时候的时间。Sout（date）

Data类中有重载的构造器，但是基本都过时了。

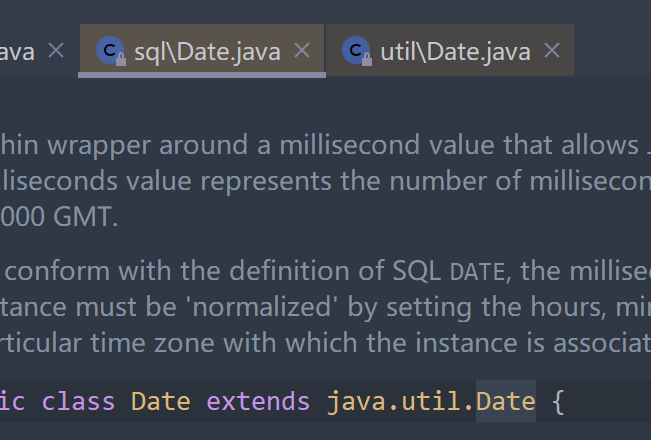
只有一个long型数值的构造器里面写从1970到现在的ms数，注意加L表示long型数值。Date（） date=new Date（xxxxxxL）；此时sout（date），输出的就是ms数转为的时间。

过时的方法可以用其他类的方法替代，在Date类中有描述。

Date对象有一个方法getTime（），也就是获取这个对象new的时候举例1970.1.1的ms数，返回一个long型数值。也是一个时间戳，跟System.currentTimeMillis()一样的效果。

## Sql包下的时间类-上

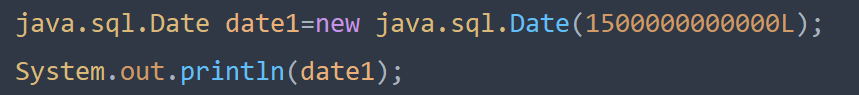
这个Date类是继承于Util包下的Date类的

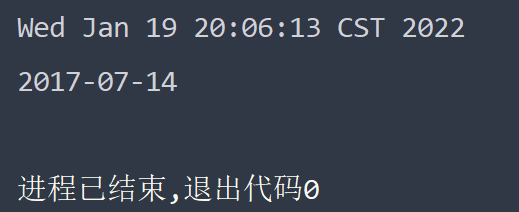


这个对象就剩一个构造器没过时了，就是往里面写long型的ms数。

Java.sql.Date date=new Java.sql.Date(xxxL)；

此时sout（date）输出的是ms数转为的年-月-日。





# 复习多态

现在复习一下父类子类的转换问题：

我们都知道子类继承父类，然后在某个方法中我们可以将形参声明为父类类型的，传入子类的实例（子类的地址），我们称之为多态。

当我们把多态从方法层面扩展到属性层面，我们可以声明父类变量，赋值子类对象。

上升到类的层面，我们可以定义一个接口，让实现类去实现它，在别的方法中我们可以用接口类型的变量，然后赋值很多很多其他的实现类。 我们可以new一个类，然后声明为他的父类类型的。

来到强转层面如果A类是B类的父类，我们可以A a=（A）b，意思是将b的对象地址重新赋值给A类型的变量a。

我们为什么能这么强转呢？因为多态，因为java就这么定义的，强转后是不是就相当于A a=new B（）？不就相当于把一个子类的对象地址赋值给父类的变量？

所以反过来B b=（B）a；这就不对了。如果把父类对象的地址给子类类型变量，那就违背了多态，那我们岂不是可以在方法形参中写一个子类变量，穿父类的地址了吗？然并卵，会报错！

那么只有一种情况能把父类的变量强转给子类类型的变量，即这个父类对象本来储存的就是子类的对象地址即

B b=new B（）； A a=（A）b； 然后 B b1=（B）a。这样就不会报错，因为地址本来就是子类的，最后再让子类变量储存子类对象地址完全可以。

**归根结底，子类类型的变量只能储存子类对象地址，父类类型的变量不仅可以存自己对象的地址，还可以储存他的子类对象的地址，只不过就是只能调父类中的方法罢了！**

## Sql包下的时间类-下

Util下的Date类date1对象有getTime（）方法返回ms数。

而sql下的Date类正好有放long型数值的构造器

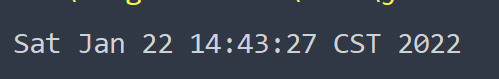
可以 java.sql.Date date2=new java.sql.Date（date1.getTime()）

即将Util下的时间类对象的getTime方法返回的ms数直接方法sql下的Date构造器中，然后诞生一个sql下的Date类对象。

就变相地将Util下Date转为sql下的Date了，在我们将java中时间类地时间数据与数据库时间类时间数据连通时使用。

把java显示的时间转为数据库显示的时间

# SimpleDateFormat类



由于默认输出的date对象是用英文表示的，方便我们转为各种格式的提供了此类辅助转换。

时间格式化类SimpleDateFormat：格式化：主要功能为将时间转为String字符串。

解析：将字符串转为时间。

首先实例化SimpleDateFormat对象，对于参数类型是date的，一般用默认构造器和带参构造器。

默认构造器不指定格式化格式，带参构造器可以传入指定格式化格式字符串：比如“yyyy-mm-dd hh:mm:ss”。

New出来对象以后，调用.format()方法，传入Date类对象date，date的作用是获取本地时间并封装到对象中，所以我们要将时间格式化为某个格式得先把这个时间对象造出来即date。

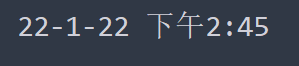
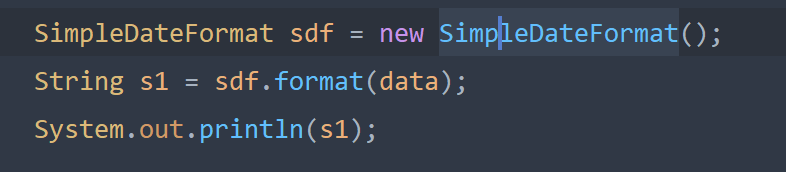
将date对象放入format方法以后，方法会返回一个字符串，用String类型的变量接收一下。

Date date=new Date();

SimpleDateFormat sdf=new SimpleDateFormat（）；

Sout.Sdf.format(date);

此时输出的就是默认格式化以后的时间字符串。

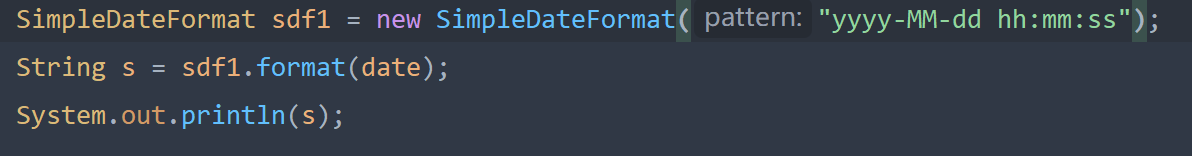


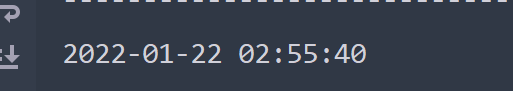
解析，如果是默认构造器new的sdf对象，那么日期格式是默认的那种，将date对象转为字符串也是默认的格式。解析就是调用sdf的parse方法，将默认格式的字符串放进去，重新new一个date对象给我们，我们用Date类的变量接收一下这个对象。

即如果是默认构造器造出来的sdf对象，时间格式也是默认的，调用format方法传入date对象出来的时间字符串也是默认提供的。如果我们调用parse方法将字符串重新封装到一个新的date对象中，这个字符串的格式必须也是默认的。否则就会报异常，使用parse方法的时候就会让处理异常，就是怕我们传入的字符串格式不是默认的。

但是SimpaleDateFormat类的带参构造器中还可以传入字符串类型的指定格式化模板。

比如newSimpleDateFormat对象构造器中写的是指定时间格式的字符串（这些指定格式在api中有描述不能乱写），然后调用formate方法传入date对象格式化出来的时间字符串就是我们在构造器中写入的格式。





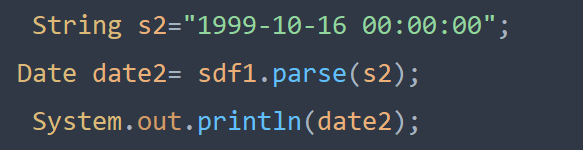
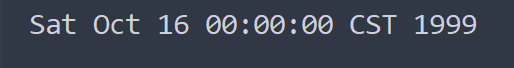
这样就变得更通俗易懂了。

同样，按照这个时间字符串将这个格式封装到date对象中，sdf对象调用parse方法解析，生成date对象，注意这个字符串是什么格式的，调用的sdf对象就得是什么格式的构造器new出来的。这样才能相匹配。

即Date对象格式永远只有一种

什么格式的构造器new出来的sdf对象，format方法传入date对象出来的字符串就是什么格式。

Sdf对象的parse方法传入的时间字符串格式必须是此sdf对象被new时构造器里面的格式，才能重新生成Date对象，并且输出此Date对象，输出的时间就是字符串我们填写的时间。

# Calendar日历类

Calendar是被abstract修饰的，是抽象类，不可实例化，必须被继承。

那么我们实际上是使用它的子类GregorianCalendar公历日历来new各种对象调用各种对象来使用各种方法。

实例化对象两种方式，一种是直接实例化GregorianCalendar对象。

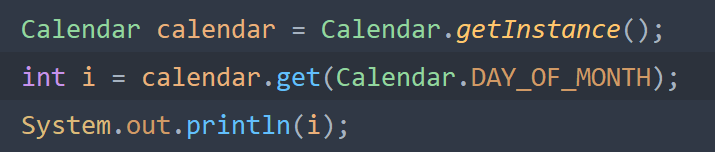
第二种，调用Calendar类中的getInstance方法，返回一个Calendar类型的子类GregorianCalendar。

所以我们必须声明一个Calendar类型的变量来接收。

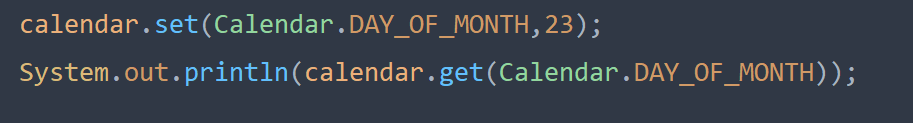
接收之后可以调用对象的get方法，传入参数来得到方法的各种返回值。

比如传入参数Calendar类的静态final属性 Calendar.DAY-OF-MONTH。

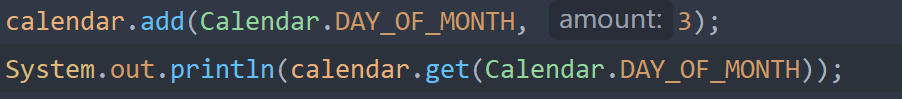
返回的就是int类型的数值，这个月的第几天



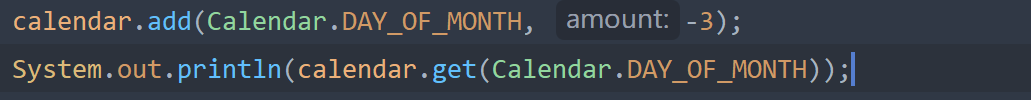
同样我们可以set属性DAY-OF-MONTH的值，之后再get就是我们set的值了。



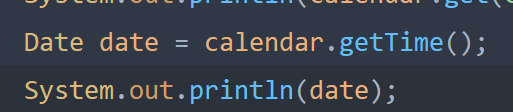
我们还可以添加天数



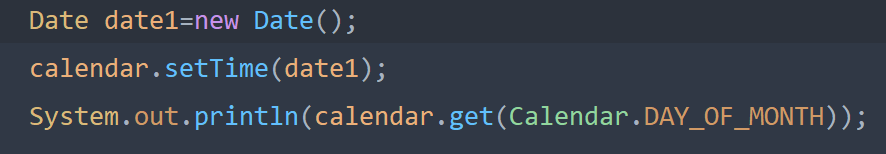
删除天数，就是加一个负数天数



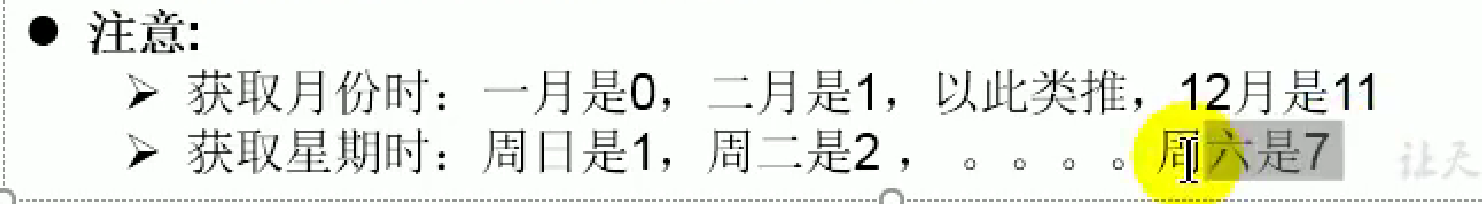
日历类可以生成对应的Date类，调用getTime方法返回的就是Date对象



setTime方法就是将一个Date对象传入，然后将date对象的时间设置为此日历类的时间。没有返回新的日历类。



我们可以将一个字符串通过SimpleDateFormat类对象封装到date对象中，然后将date对象通过setTime方法封装到日历类对象中，再调用日历类对象的get方法得到这个日期是这一年的第几天啥的。



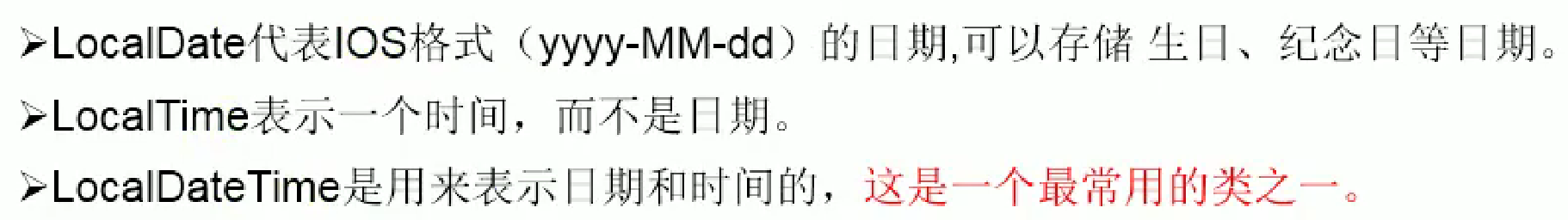
因为是循环取余！

# JDK8中新的LocalDate，LocalDateTime，LocaTime类

Date类缺点

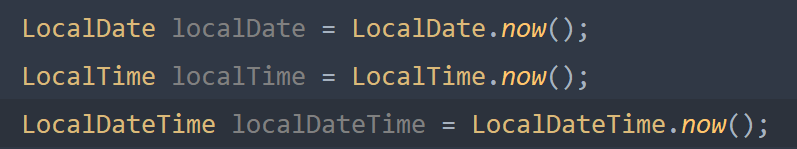


新的常用类



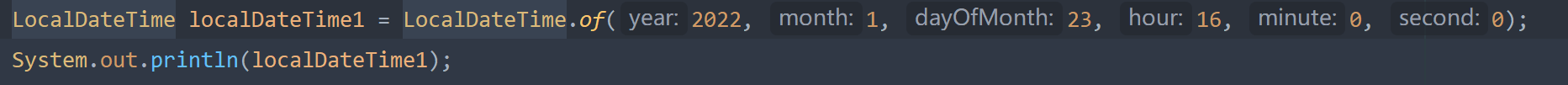
## 实例化

LocalDateTime，LocalDate，LocalTime，这三个类的now方法是静态的，方法返回的就是当前类的对象。



其中时间是精确到ms的

拿LocalDateTime举例，我们可以通过他的静态方法of（），填入指定的年月日，时分秒啥的，生成新的LocalDateTime对象。



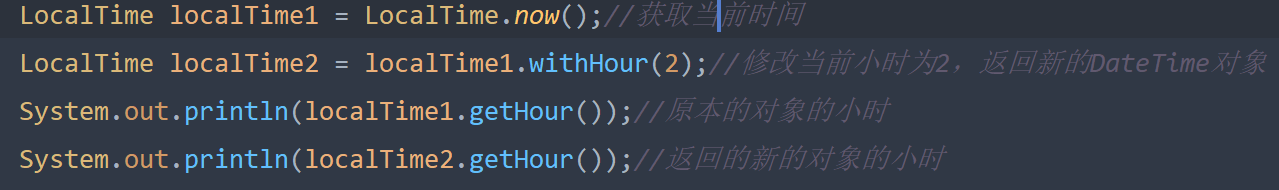


此时不再有偏移量问题。

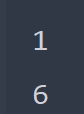
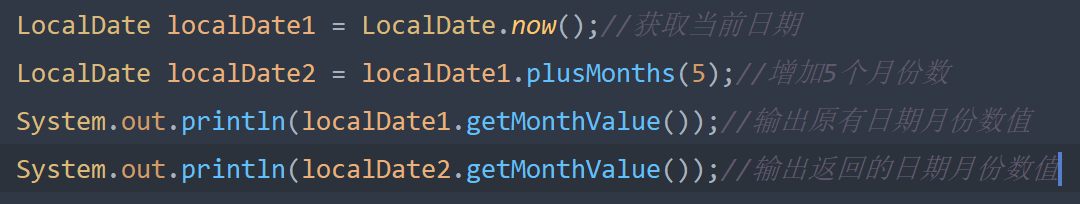
## getxxx方法



## withxxx方法设置值—不可变性

可以看到原本的locatime对象是没有被修改的而是返回了修改后的新的localtime对象。

## plusxxx方法增加值—不可变性

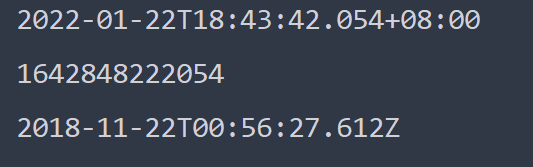


类似于JDK1.1中的Calendar类

# JDK8中新的Instant类瞬时类

类似于Date类

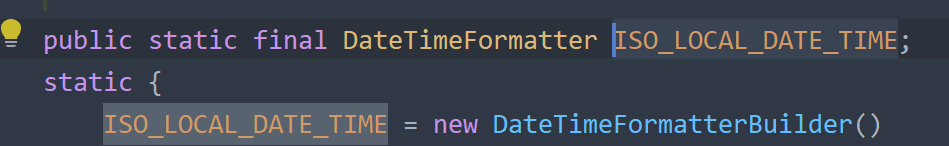




# JDK8 xxxFormatter类 格式化，解析类

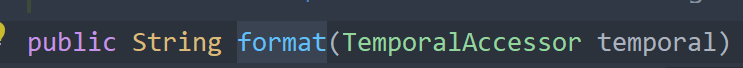
DateTimeFormatter类为例，类似于SimpleDateFormat类。

## 实例化

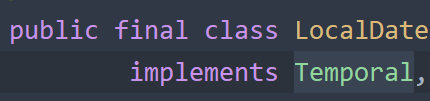
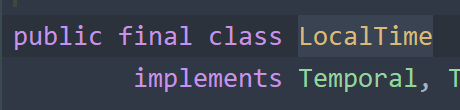
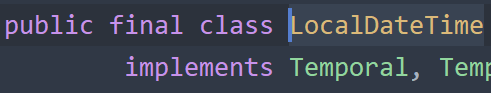


DateTimeFormatter dtf=DateTimeFormatter.ISO\_LOCAL\_TIME；

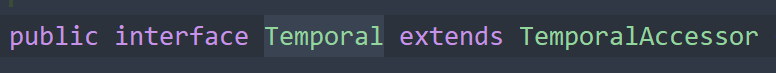
说明此类是帮助格式化默认的LocalDateTime日期格式的。

这个类有个format方法

看参数，不是LocalDateTime的，而是一个接口，说明要传接口的实现类，或者接口的子接口的实现类。

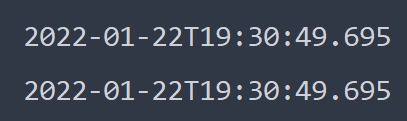
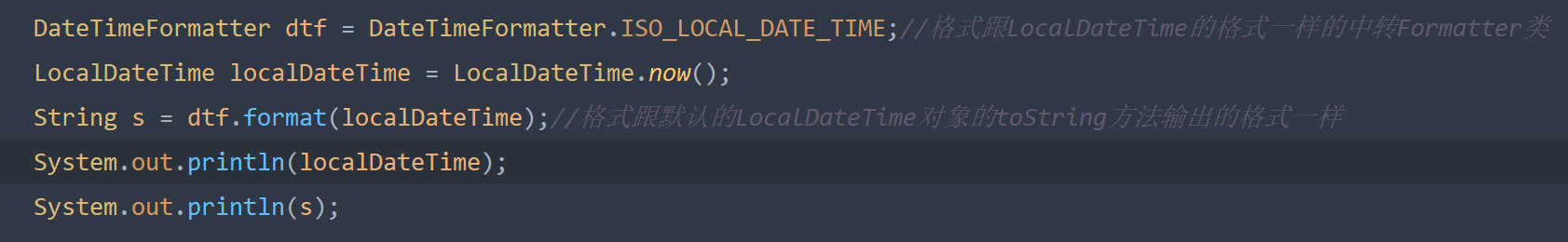


这三个类实现了接口Temporal。



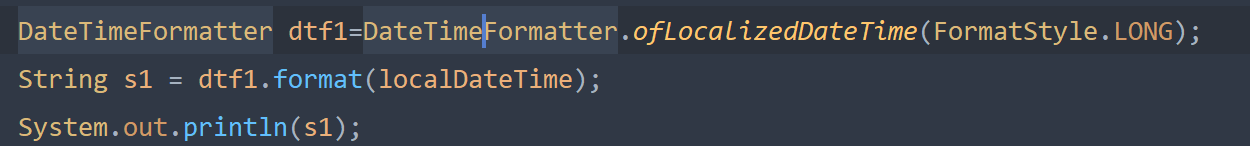
这个接口的父接口就是TemporalAccessor

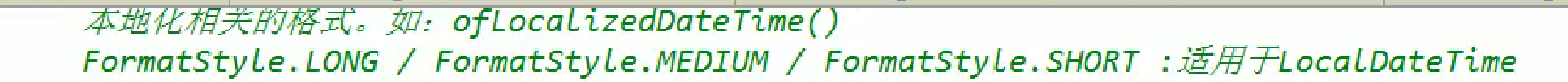
说明我们可以直接将LocalDateTime LocalTime LocalDate三个类的对象传入foramt方法中。格式化成一个时间字符串并返回。



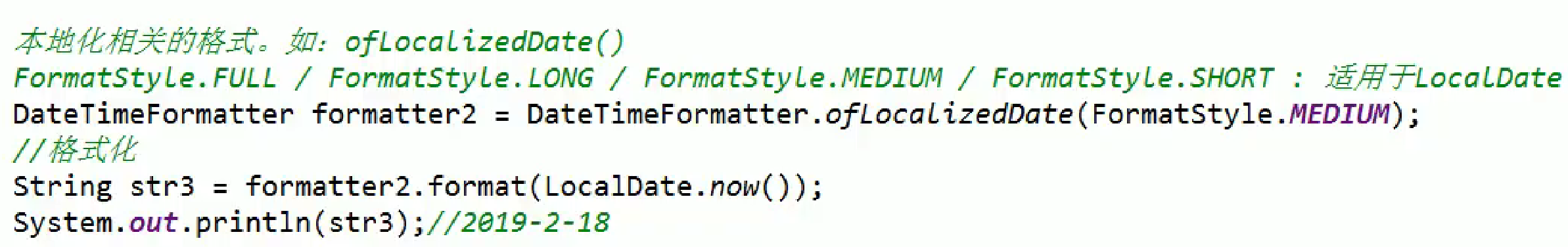
## 格式

除了DateTimeFormatter构造器中指定LocalDateTime默认格式的，还可以指定本地格式的。比如格式化的字符串中带有上午下午的字样啥的。





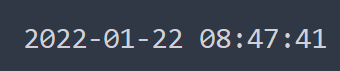
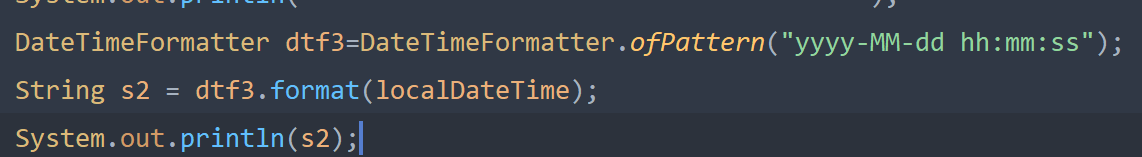
解析回去就还是调用parse（传入相应2022年1月22日 下午08时30分00秒）这样的。



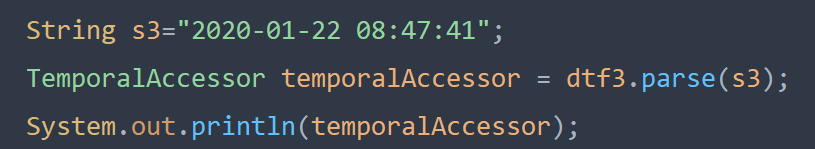
## 自定义的时间格式



格式化



解析





# JDK8其他关于时间的API



