Date

`java.util.Date` 表示特定的时间（某一个瞬间），用来表示时间和日期，提供一系列操作。

```
public class Date
    implements java.io.Serializable, Cloneable, Comparable<Date>
```

- 获取的是系统的时间和日期
- 1天= 24×60×60 = 86400秒
- 计算机标准是根据Greenwich(格林威治)标准时间（GMT），由于中国属于东八区，所以要比 `GMT` 时间早8个小时

类的所有方法 `Date` 接受或返回年，月，日，小时，分钟和秒值，以下表述中使用：

- `y` 年代表整数 y `- 1900` 。
- 一个月由0到11的整数表示; 0是1月，1是2月，等等，11是12月
- 日期（月的一天）以通常的方式从1到31的整数表示。
- 一小时由0到23之间的整数表示。因此，从午夜到凌晨1点的时间是小时0，从中午到下午1点的小时是12小时。
- 一般以0～59的整数表示 `minute` 。
- 秒由0到61的整数表示; 值60和61仅发生在闰秒上，甚至仅在实际上正确跟踪闰秒的Java实现中发生。 由于目前引入闰秒的方式，在同一分钟内不会发生两个闰秒，但是本规范遵循ISO C的日期和时间约定。

在所有情况下，为这些目的而提供的方法的论证不必在指定范围内; 例如，可以将日期指定为1月32日，并将其解释为2月1日

成员变量

```
private transient long fastTime;// 保存Date所表示的时间
```

构造方法

目前还有两个未被弃用的构造方法：

Constructor	Description
`Date()`	分配一个 `Date`对象，并初始化它，当前系统时间

Constructor	Description
Date(long date)	分配一个 Date对象，date毫秒。即1970年1月1日00:00:00 GMT以来的毫秒数

```
public Date(long millis) {
    fastTime = millis ;
}

public Date() {
    this(System.currentTimeMillis());//将System.currentTimeMillis()保存到fastTime
}
```

`System.currentTimeMillis()` 也是获取自1970年1月1日以来，以此日期为准的00:00:00 GMT的毫秒数。

```
public static void main(String[] args) {
    Date date = new Date();
    System.out.println(date);

    Date date1 = new Date(1000L);
    System.out.println(date1);
}
```

注意： `CST` 表示 `China Standard Time` (中国标准时间)

成员方法

`Date` 中大部分方法都已经经被弃用，仅存在以下成员方法：

Method	Type	Description
after(Date when)	boolean	测试此日期是否在指定日期之后
before(Date when)	boolean	测试此日期是否在指定日期之前
equals(Object obj)	boolean	比较两个日期相等
getTime()	long	自1970年1月1日以来，以此日期为准的00:00:00 GMT的毫秒数

Method	Type	Description
`setTime(long time)`	`void`	设置此 Date对象以表示1970年1月1日00:00:00 GMT后的time毫秒的时间点
`toString()`	`String`	将此 `Date`对象转换为 `String`的形式

其中 `toString()` 的格式是： `dow mon dd hh:mm:ss zzz yyyy`

- `dow(day of week)` 是星期几（ `Sun, Mon, Tue, Wed, Thu, Fri, Sat` ）。
- `mon` 是月（ `Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec` ）。
- `dd` 是一个月的某天（ `01` 到 `31` ），作为两位十进制数字。
- `hh` 是一天的小时（ `00` 到 `23` ），作为两位十进制数字。
- `mm` 是一小时内的分钟（ `00` 至 `59` ），为两位十进制数字。
- `ss` 是分钟内的秒（ `00` 到 `61` ），作为两位十进制数字。
- `zzz` 是时区。 标准时区缩写包括通过方法 `parse识别的缩写` 。 如果时区信息不可用，那么 `zzz` 是空的 - 也就是说，它根本没有字符。
- `年份` 为 `yyyy` ，为四位十进制数字

比较Date对象

`Date` 对象的比较可以通过 `equals` 、 `after` 、 `before` 方法实现

```
public static void main(String[] args) {
    long mills = System.currentTimeMillis(); //获取当前时间的毫秒值
    Date first = new Date(mills);
    System.out.println(first);

    Date second = new Date(mills);
    System.out.println(second);
    // == 比较
    System.out.println(first == second);//false
    //equals比较，Date重写了从Object继承的equals方法
    //内部实现是 obj instanceof Date && getTime() == ((Date) obj).getTime()
    System.out.println(first.equals(second));//true

    Date now = new Date();

    System.out.println(now.after(first));//true
    System.out.println(now.after(second));//true

    System.out.println(first.before(now));//true
    System.out.println(second.before(now));//true
}
```

Date实例是可变的

`Date` 实例（对象）创建后，内部的毫秒值 `fastTime` 是可以改变的。

```
public static void main(String[] args) {

    long ms = 1000L * 60 * 60 * 24 * 365 * 25 ;//1995

    final Date birthdate = new Date(ms);
    System.out.println(birthdate);//Sat Dec 25 08:00:00 CST 1999

    birthdate.setTime(1000);//改变Date对象
    System.out.println(birthdate);//Thu Jan 01 08:00:01 CST 1970
}
```

Calendar

`java.util.Calendar` 是一个抽象类。可以为在某一特定时刻和日历字段之间的转换的方法，以及用于操纵该日历的字段提供了方法， 时间上的瞬间可以用毫秒值表示，该值是从1970年1月1日00:00 00：00.000 GMT

```
public abstract class Calendar implements Serializable, Cloneable,
Comparable<Calendar>
```

由于该类是抽象类，所以不能直接实例化，在类中提供了一个方法 `Calendar.getInstance();` ,这个方法返回了一个 `Calendar` 对象，其日历字段已使用当前日期和时间进行初始化。

类变量

- `Calendar.ERA` 对应纪元

  比如罗马儒略历中的 AD 或 BC (中文表示为 公元 或 公元前)

- `Calendar.YEAR` 对应年份
- `Calendar.MONTH` 对应月份

  月份从 0 开始计数，0 表示 `January` (一月)，1 表示 `February` (二月)

- `Calendar.DATE` 对应月份中的日期

日期从 1 开始计数，有效范围为 1 ~ 31 。

- `Calendar.HOUR_OF_DAY` 对应一天当中的小时

  小时从 0 开始计数，有效范围为 0 ~ 23 。

- `Calendar.MINUTE` 对应分钟

  分钟从 0 开始计数，有效范围为 0 ~ 59 。

- `Calendar.SECOND` 对应秒

  秒数从 0 开始计数，有效范围为 0 ~ 60 。(要注意闰秒)

- `Calendar.MILLISECOND` 对应毫秒

  毫秒数从 0 开始计数，有效范围为 0 ~ 999 。

- `Calendar.DAY_OF_MONTH` 对应月份中的日期

  日期从 1 开始计数，有效范围为 1 ~ 31 。

- `Calendar.HOUR` 对应上午或下午的小时

  小时从 0 开始计数，有效范围为 0 ~ 11 。

- `Calendar.DAY_OF_WEEK` 对应星期

  用于指示一个星期中的某天。

  该字段可取的值可以是 `SUNDAY` 、 `MONDAY` 、 `TUESDAY` 、 `WEDNESDAY` 、 `THURSDAY` 、 `FRIDAY` 和 `SATURDAY` 。

- `Calendar.DAY_OF_YEAR` 对应年份中的天数

  指示当前年中的天数。一年中第一天的值为 1，最大值为 366 。

- `Calendar.AM` 表示一天午夜到中午之前的时间，即上午

- `Calendar.PM` 表示从中午到午夜之前的一天中的一段时间，即下午

获取Calendar对象

通过子类获取

`java.util.GregorianCalendar` 是 `Calendar` 类的子类，并提供了世界上大多数国家使用的标准日历系统。通过 `GregorianCalendar` 对象即可使用 `Calendar` 中的方法

```
Calendar calendar = new GregorianCalendar();
System.out.println(calendar);
```

通过Calendar的static方法获取

`Calendar` 类中提供了大量的 `static` 方法用于获取 `Calendar` 对象

举例

```
Calendar c = Calendar.getInstance(); //使用默认时区和区域设置获取日历
Calendar c1 = Calendar.getInstance(Locale.CHINA);//使用默认时区和指定的区域设置获取日历
Calendar.getInstance(TimeZone.getDefault());//使用指定的时区和默认语言环境获取日历
Calendar.getInstance(TimeZone.getDefault(), Locale.CHINA);//获取具有指定时区和区域设置的日历
```

成员方法

获取时间

- 获取 `Calendar` 对象所表示的某个特定瞬间所对应的毫秒值

```
public long getTimeInMillis();
```

- 获取 `Calendar` 对象所表示的某个特定瞬间的 `指定字段` 所对应的值

```
public int get(int field);//field是前面提到的Calendar的类变量
```

```
package com.itlaobing.note;

import java.util.Calendar;
import java.util.GregorianCalendar;

public class CalendarTest {

    public static void main(String[] args) {
        // 父类引用 指向 子类对象
        Calendar c = new GregorianCalendar();
```

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

        // 从 c 对应的 Calendar 实例中获取 日历字段 ERA 对应的值
        int era = c.get( Calendar.ERA ) ;
        System.out.println( era );//1

        int year = c.get( Calendar.YEAR );
        System.out.println( year );//2019

        // 在西方国家，月份的索引从 零 开始，0 表示 1月 ，  11 表示12月
        int month = c.get( Calendar.MONTH );
        System.out.println( month ); //9

        //对应月份中的日期
        int date = c.get( Calendar.DATE );
        System.out.println( date );//24

        //一天中的小时 24时制
        int hourOfDay = c.get( Calendar.HOUR_OF_DAY );
        System.out.println( hourOfDay );//20

        //分钟
        int minute = c.get( Calendar.MINUTE );
        System.out.println( minute );//0

        //秒
        int second = c.get( Calendar.SECOND );
        System.out.println( second );//13

        // 仅仅获取 MILLISECOND 字段对应的值，取值范围为 [ 0 , 1000 )
        int millis = c.get( Calendar.MILLISECOND );
        System.out.println( millis );//48

        System.out.println( "~ ~ ~ ~ ~ ~ ~ ~ ~ ~" );

        // 获取 当前的 Calendar 实例所表示的瞬间 对应的毫秒值(距离历元的偏移量)
        long ms = c.getTimeInMillis();
        System.out.println( ms );//1571918413048
    }

}

```

设置/清除Calendar对象值

`Calendar`类中提供了一下方法用于设置`日历字段`的值

```
public void set(int field, int value)
```

同时也提供了一下方法来批量设置`日历字段`值

```
public final void set(int year, int month, int date);

public final void set(int year, int month, int date, int hourOfDay, int minute);

public final void set(int year, int month, int date, int hourOfDay, int minute, int second)
```

另外，`Calendar`类中提供了清空指定`日历字段`值的方法

```
public final void clear(int field);
```

`Calendar`类中还提供了清空所有`日历字段`值的方法

```
public final void clear();
```

举例

```
package com.itlaobing.note;

import java.util.Calendar;
import java.util.GregorianCalendar;

public class CalendarTest2 {

    public static void main(String[] args) {
        // 父类引用 指向 子类对象
        Calendar c = new GregorianCalendar();
        System.out.println(c);

        c.set(Calendar.YEAR, 1995); // 公元1995年
        c.set(Calendar.MONTH, 0); // 月份的索引从 0 开始, 0表示1月
        c.set(Calendar.DATE, 10);
        c.set(Calendar.HOUR_OF_DAY, 11);
        c.set(Calendar.MINUTE, 45);
        c.set(Calendar.SECOND, 0);
        c.set(Calendar.MILLISECOND, 0);
        System.out.println(c);

        System.out.println(c.getTimeInMillis());
        System.out.println(c);

        System.out.println("============================");

        c.clear(); // 清除所有日历字段的值
        System.out.println(c);
```

```
29
30          // YEAR 、MONTH 、HOUR_OF_DAY 、MINUTE、SECOND
31          c.set(0, 0, 10, 11, 45, 30);
32          c.set(Calendar.MILLISECOND, 100);
33
34          System.out.println(c.getTimeInMillis());//-62166600869900
35          System.out.println(c);
36      }
37
38  }
39
```

其他方法

在某个字段得基础上添加

```
1  add(int field, int amount)
```

判断两个日历对象的顺序

```
1  boolean after(Object when)
2  boolean before(Object when)
```

Calendar对象和Date对象互相转换

`Calendar`类中提供了将`Calendar`对象所表示的某一特定瞬间转换成`Date`对象的方法

```
1  public final Date getTime();
```

`Calendar`类中提供了将`Date`对象转换成`Calendar`对象所表示的某一特定瞬间的方法

```
1  public final void setTime(Date date);
```

举例

```
1  package com.itlaobing.note;
2
```

```java
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;

public class CalendarTest3 {

    public static void main(String[] args) {
        Calendar calendar = new GregorianCalendar();
        // java.util.Date 类中的 getTime() 方法可以返回一个毫秒值
        // java.util.Calendar 类中的 getTime() 用于返回相应的 Date 实例
        Date d = calendar.getTime();
        System.out.println(d);

        Date date = new Date();
        Calendar c = Calendar.getInstance();
        c.clear(); //清空所有字段值
        c.setTime(date); //设置时间
        System.out.println(c);
    }

}

```

DateFormat

`java.text.DateFormat` 是日期/时间格式化子类的抽象类。它可以将日期时间格式化和将字符串解析为 `Date`

```java
public final String format(Date date);//格式化
public Date parse(String source);//解析字符串为Date
```

但是， `DateFormat` 是一个抽象类。所以使用时使用其子类 `SimpleDateFormat`

`SimpleDateFormat` 的构造方法

```java
public SimpleDateFormat();

public SimpleDateFormat(String pattern);

public SimpleDateFormat(String pattern, Locale locale);

public SimpleDateFormat(String pattern, DateFormatSymbols formatSymbols);
```

最常用的是 `public SimpleDateFormat(String pattern)` ，其中 `pattern` 表示日期的格式，日期格式规则如下：

字母	日期或时间元素	表示	示例
G	Era 标志符	Text	AD
y	年	Year	1996; 96
M	年中的月份	Month	July; Jul; 07
w	年中的周数	Number	27
W	月份中的周数	Number	2
D	年中的天数	Number	189
d	月份中的天数	Number	10
F	月份中的星期	Number	2
E	星期中的天数	Text	Tuesday; Tue
a	Am/pm 标记	Text	PM
H	一天中的小时数（0-23）	Number	0
k	一天中的小时数（1-24）	Number	24
K	am/pm 中的小时数（0-11）	Number	0
h	am/pm 中的小时数（1-12）	Number	12
m	小时中的分钟数	Number	30
s	分钟中的秒数	Number	55
S	毫秒数	Number	978
z	时区	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	时区	RFC 822 time zone	-0800

比如 `yyyy-MM-dd` 对应 `1995-01-01`

Date格式化

```java
package com.itlaobing.note;

import java.text.SimpleDateFormat;
import java.util.Date;

public class SimpleDateFormatTest {

    public static void main(String[] args) {
        //重点:  将java.util.Date对象表示的【瞬间】格式化为【指定格式】的字符串
        long ms = 1000L * 60 * 60 * 24 * 365 * 26 ;

        Date birthdate = new Date(ms);
        System.out.println(birthdate);//Tue Dec 26 08:00:00 CST 1995

        String pattern = "G yyyy年MM月dd日 EEEE HH:mm:ss.SSS" ; // 确定 "日期时间" 的格式
        SimpleDateFormat sdf = new SimpleDateFormat(pattern);

        String str = sdf.format(birthdate);
        System.out.println(str);//公元 1995年12月26日 星期二 08:00:00.000
    }
```

```
21
22  }
23
```

将字符串解析为Date对象

将指定字符串按照特定的格式解析为 `Date` 对象:

```
1  package com.itlaobing.note;
2
3  import java.text.ParseException;
4  import java.text.SimpleDateFormat;
5  import java.util.Date;
6  import java.util.Scanner;
7
8  public class SimpleDateFormatTest2 {
9
10     public static void main(String[] args) {
11         //重点: 将 指定格式字符串 解析为 java.util.Date 对象
12
13         final String pattern = "yyyy-MM-dd";//字符串格式
14         SimpleDateFormat sdf = new SimpleDateFormat(pattern);
15
16         Scanner sc = new Scanner(System.in);// 创建扫描器读取用户输入的数据
17
18         System.out.println("请输入一个日期( 格式为 " + pattern + " , 比如 2019-10-24
   )");
19         String s = sc.nextLine(); // 读取用户输入的整行数据
20         System.out.println("你输入的是: " + s); //你输入的是: 2019-10-10
21
22         try {
23             // 将 字符串 解析为 Date 类型的实例
24             Date date = sdf.parse(s); //如果格式不正确, 可能抛出 ParseException 异常
25             System.out.println(date);//Thu Oct 10 00:00:00 CST 2019
26             long time = date.getTime();//1570636800000
27             System.out.println(time);
28         } catch (ParseException e) {
29             System.out.println("你输入的日期格式不符合 " + pattern);
30             e.printStackTrace();
31         }
32         sc.close(); // 关闭扫描器
33     }
34 }
35
```

JDK 8新增

LocalDate

`java.time.LocalDate` 是一个不可变(`final` 修饰)的日期时间对象,是线程安全的，表示日期，通常被表示为年月日。也可以访问其他日期字段，例如日期，星期几和星期

```
public final class LocalDate
        implements Temporal, TemporalAdjuster, ChronoLocalDate, Serializable {
```

字段

`LocalDate` 中声明了三个字段来存储年、月、日

```
/**
 * The year.
 */
private final int year;
/**
 * The month-of-year.
 */
private final short month;
/**
 * The day-of-month.
 */
private final short day;
```

因为它们都是 `final` 修饰的，因此一旦创建 `LocalDate` 实例，其 年份、月份、日期 的值再也不能被更改。

另外， `LocalDate` 类还声明了以下常量:

```
public static final LocalDate MIN = LocalDate.of( Year.MIN_VALUE, 1, 1
);//-999999999-01-01
public static final LocalDate MAX = LocalDate.of( Year.MAX_VALUE, 12, 31
);//+999999999-12-31
```

`JDK 9` 及之后版本还声明了以下常量

```
public static final LocalDate EPOCH = LocalDate.of( 1970, 1, 1 );
```

表示 `记元` 对应的日期

获取LocalDate实例

`java.time.LocalDate` 类提供了许多`类方法`用于获取 `LocalDate` 实例

```
public static LocalDate now();
public static LocalDate now(ZoneId zone);
public static LocalDate now(Clock clock);

public static LocalDate of(int year, Month month, int dayOfMonth);
public static LocalDate of(int year, int month, int dayOfMonth);
public static LocalDate ofYearDay(int year, int dayOfYear);
public static LocalDate ofEpochDay(long epochDay);
```

常用的是`now()`、`of(int year , int month , int dayOfMonth)`、`ofYearDay(int year , int dayOfYear)`

举例

```
package com.itlaobing.note;

import java.time.LocalDate;

public class LocalDateTest {

    public static void main(String[] args) {
        // 通过 LocalDate 类中的 类方法 ( now ) 来获取 LocalDate 实例
        LocalDate now = LocalDate.now();//初始化值为当前系统时间
        System.out.println(now);

        // 通过 LocalDate 类中的 类方法 ( of ) 来获取 LocalDate 实例
        LocalDate birthdate = LocalDate.of(1995, 9, 14);
        System.out.println(birthdate);

        // 通过 LocalDate 类中的 类方法 ( ofYearDay ) 来获取 LocalDate 实例
        LocalDate date = LocalDate.ofYearDay(2000, 297);
        System.out.println(date);
    }

}

```

常用实例方法

`LocalDate` 类中定义了大量的实例方法，其中比较常用的是:

- `public int getYear()` 用于获取年份
- `public int getMonthValue()` 用于获取月份 (取值范围是 1 ~ 12)
- `public int getDayOfMonth()` 用于获取日期 (取值范围是 1 ~ 31)
- `public DayOfWeek getDayOfWeek()` 用于获取星期 (返回类型为 `DayOfWeek`)
- `public boolean isLeapYear()` 用于判断 `LocalDate` 实例对应的年份是否是闰年
- `public int lengthOfYear()` 用于获取 `LocalDate` 实例对应的年份的总天数
- `public int lengthOfMonth()` 用于获取 `LocalDate` 实例对应的月份的总天数

举例

```
package com.itlaobing.note;

import java.time.DayOfWeek;
import java.time.LocalDate;

public class LocalDateTest2 {

    public static void main(String[] args) {
        LocalDate ld = LocalDate.now();

//      ld = LocalDate.of(2008, Month.AUGUST, 8);
        ld = LocalDate.of(2008, 8, 8);

        ld = LocalDate.ofYearDay(2002, 222);

        System.out.println(ld);


        int y = ld.getYear();
        Month m = ld.getMonth();
        int m1 = ld.getMonthValue();
        int doy = ld.getDayOfYear();
        int d = ld.getDayOfMonth();

        DayOfWeek w = ld.getDayOfWeek();
        boolean isLeapYear = ld.isLeapYear();

        int days = ld.lengthOfYear();
        int ds = ld.lengthOfMonth();

        System.out.println(ds);

}

```

`LocalDate`类中提供了`LocalDate`对象之间相互比较的方法

```
public boolean isEqual(ChronoLocalDate other);
public boolean equals(Object obj);
public boolean isAfter(ChronoLocalDate other);
public boolean isBefore(ChronoLocalDate other);
```

同时，`LocalDate`类中还提供了在指定`LocalDate`基础上增加/减少指定时间的方法

```
//增加
public LocalDate plusYears(long yearsToAdd);
public LocalDate plusDays(long daysToAdd);
public LocalDate plusMonths(long monthsToAdd);
public LocalDate plusWeeks(long weeksToAdd);
//减少
public LocalDate minusYears(long yearsToSubtract);
public LocalDate minusMonths(long monthsToSubtract);
public LocalDate minusWeeks(long weeksToSubtract);
public LocalDate minusDays(long daysToSubtract);
```

如果调用增加方法传负数的话就相当于减少，所以一般都是使用增加方法。

举例

```
package com.itlaobing.note;

import java.time.LocalDate;

public class LocalDateTest3 {

    public static void main(String[] args) {
        LocalDate date = LocalDate.now();
        System.out.println(date);

        LocalDate plusYear = date.plusYears(5);

        // 10 年前
        LocalDate minusYear = date.minusYears(10);
        LocalDate minusYear1 = date.plusYears(-10);


        LocalDate plusMonth = date.plusMonths(5);

        // 5 月前
        LocalDate minusMonth = date.minusMonths(5);
        LocalDate minusMonth1 = date.plusMonths(-5);


```

```
        // 几周后
        LocalDate plusWeek = date.plusWeeks(1);

        LocalDate minusWeek = date.minusWeeks(1);
        LocalDate minusWeek1 = date.plusWeeks(-1);


        System.out.println(minusWeek.compareTo(date));
    }

}

```

LocalTime

`java.time.LocalTime` 是一个不可变(`final`)的日期时间对象，代表一个时间。 时间表示为纳秒精度。它不存储或表示日期或时区。

```
public final class LocalTime
        implements Temporal, TemporalAdjuster, Comparable<LocalTime>, Serializable
```

在 `java.time.LocalTime` 类中声明了四个字段用来存储 小时、分钟、秒、纳秒:

```
    /**
     * The hour.
     */
    private final byte hour;
    /**
     * The minute.
     */
    private final byte minute;
    /**
     * The second.
     */
    private final byte second;
    /**
     * The nanosecond.
     */
    private final int nano;
```

因为它们都是 `final` 修饰的，因此一旦创建 `LocalTime` 实例，其中各个字段的值再也不能被更改

获取 LocalTime 实例

`java.time.LocalTime` 类提供了许多 `类方法` 用于获取 `LocalTime` 实例，以下是几个常用的方法

```
public static LocalTime now()
public static LocalTime of(int hour, int minute)
public static LocalTime of(int hour, int minute, int second)
public static LocalTime of(int hour, int minute, int second, int nanoOfSecond)
public static LocalTime ofNanoOfDay(long nanoOfDay)
public static LocalTime ofSecondOfDay(long secondOfDay)
```

举例

```
package com.itlaobing.note;

import java.time.LocalTime;

public class LocalTimeTest {

    public static void main(String[] args) {
        LocalTime now = LocalTime.now();// 当前时间
        System.out.println(now);//21:27:03.972

        LocalTime f = LocalTime.of(12, 0);
        System.out.println(f);//12:00

        LocalTime s = LocalTime.of(12, 24, 36);
        System.out.println(s);//12:24:36

        // 1s = 1000ms (毫秒)
        // 1ms = 1000us (微秒)
        // 1us = 1000ns ( 纳秒 或 毫微秒 )
        LocalTime t = LocalTime.of(12, 24, 36, 100200300);
        System.out.println(t);//12:24:36.100200300
    }

}

```

常用的成员方法

`LocalTime` 类中定义了大量的实例方法，其中比较常用的是:

- `public int getHour()` 用于获取小时 (取值范围为 0 ~ 23)
- `public int getMinute()` 用于获取分钟 (取值范围为 0 ~ 59)
- `public int getSecond()` 用于获取秒 (取值范围为 0 ~ 59)

- `public int getNano()` 用于获取纳秒 (取值范围为 0 ~ 999999999)

举例

```
package com.itlaobing.note;

import java.time.LocalTime;

public class LocalTimeTest2 {

    public static void main(String[] args) {
        LocalTime now = LocalTime.now();
        System.out.println(now);//21:29:24.003

        System.out.println(now.getHour() + "时");//21时
        System.out.println(now.getMinute() + "分");//29分
        System.out.println(now.getSecond() + "秒");//24秒

        long nano = now.getNano(); // 其返回值为 0 ~ 999999999
        long ms = nano / 1000000; // 求取毫秒部分的数值( 范围为 0 ~ 999 )
        System.out.println(ms + "毫秒");//3毫秒
        long us = nano % 1000000 / 1000; // 求取微秒部分的数值( 范围为 0 ~ 999 )
        System.out.println(us + "微秒");//0微秒
        long ns = nano % 1000000 % 1000; // 求取纳秒部分的数值( 范围为 0 ~ 999 )
        System.out.println(ns + "纳秒");//0纳秒
    }

}

```

同时，`LocalTime` 类中定义了用于比较 `LocalTime` 实例的方法:

```
public boolean isAfter(LocalTime other);
public boolean isBefore(LocalTime other);
public boolean equals(Object obj);
```

另外，`LocalTime` 类还定义了 在指定的 `LocalTime` 对应的时间基础上 增加或减少 指定时间的方法:

```
//增加
public LocalTime plusHours(long hoursToAdd);
public LocalTime plusMinutes(long minutesToAdd);
public LocalTime plusSeconds(long secondstoAdd);
public LocalTime plusNanos(long nanosToAdd);
```

增加的方法中参数传入负数就等同于减少指定时间，所以在此不再描述减少方法

举例

```java
package com.itlaobing.note;

import java.time.LocalTime;

public class LocalTimeTest3 {

    public static void main(String[] args) {
        LocalTime time = LocalTime.now(); // 获取当前时间
        System.out.println(time);//21:33:50.752

        // 在 time 所表示的时间基础上增加 5 小时后返回新的 LocalTime 实例
        LocalTime t1 = time.plusHours(5);
        System.out.println(t1);//02:33:50.752

        // 在 time 所表示的时间基础上减少 20 分钟 后返回新的 LocalTime 实例
        LocalTime t2 = time.plusMinutes(-20);
        System.out.println(t2);//21:13:50.752

        // 在 time 所表示的时间基础上增加 30 秒 后返回新的 LocalTime 实例
        LocalTime t3 = time.plusSeconds(30);
        System.out.println(t3);//21:34:20.752

        // 在 time 所表示的时间基础上增加 100200300 纳秒 后返回新的 LocalTime 实例
        LocalTime t4 = time.plusNanos(100200300L);
        System.out.println(t4);//21:33:50.852200300

        System.out.println("=================");

        System.out.println(t1.isAfter(t2));//false
        System.out.println(t2.isBefore(t3));//true
        System.out.println(t3.equals(t4));//false
    }

}

```

LocalDateTime

`java.time.LocalDateTime`是一个不可变的日期时间对象，代表日期时间。

```java
public final class LocalDateTime
        implements Temporal, TemporalAdjuster, ChronoLocalDateTime<LocalDate>,
Serializable {
```

在 `java.time.LocalDateTime` 类中声明了两个实例字段用来存储 日期 和 时间：

```
    /**
     * The date part.
     */
    private final LocalDate date;
    /**
     * The time part.
     */
    private final LocalTime time;
```

可以看出其实 `LocalDateTime` 底层就是一个 `LocalDate` + `LocalTime` 。

这里需要注意，不仅仅 `date` 和 `time` 字段是 `final` 修饰的， `LocalDate` 、 `LocalTime` 类中的实例字段也是 `final` 修饰的，因此 `LocalDateTime` 的实例一经创建，其内部的各项取值都是不可更改的。

获取LocalDateTime实例

`java.time.LocalDateTime` 中提供了很多获取 `LocalDateTime` 实例的方法，其中常用的有：

```
public static LocalDateTime now();
public static LocalDateTime of(int year, int month, int dayOfMonth, int hour, int minute);
public static LocalDateTime of(int year, int month, int dayOfMonth, int hour, int minute, int second);
public static LocalDateTime of(int year, int month, int dayOfMonth, int hour, int minute, int second, int nanoOfSecond);
public static LocalDateTime of(LocalDate date, LocalTime time);
```

举例

```
package com.itlaobing.note;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;

public class LocalDateTimeTest {

    public static void main(String[] args) {
        LocalDateTime datetime = LocalDateTime.now();
        System.out.println(datetime);

        LocalDateTime first = LocalDateTime.of(1995, 7, 4, 7, 30);
        System.out.println(first);//1995-07-04T07:30

```

```
        LocalDateTime second = LocalDateTime.of(1999, 11, 20, 6, 30, 7);
        System.out.println(second);//1999-11-20T06:30:07

        LocalDateTime third = LocalDateTime.of(1999, 11, 20, 6, 30, 7, 0);
        System.out.println(third);//1999-11-20T06:30:07

        LocalDate date = LocalDate.of(1997, 7, 1);
        LocalTime time = LocalTime.of(10, 20);
        LocalDateTime fourth = LocalDateTime.of(date, time);
        System.out.println(fourth);//1997-07-01T10:20
    }

}
```

获取日期时间值

从 `LocalDateTime` 实例中获取 年份、月份、日期 、星期 、小时、分钟、秒、纳秒举例

```
package com.itlaobing.note;

import java.time.DayOfWeek;
import java.time.LocalDateTime;

public class LocalDateTimeTest2 {

    public static void main(String[] args) {

        LocalDateTime n = LocalDateTime.now();

        System.out.println(n.getYear() + "年");
        System.out.println(n.getMonthValue() + "月");
        System.out.println(n.getDayOfMonth() + "日");

        System.out.println(n.getDayOfWeek());//THURSDAY

        int hour = n.getHour();
        int minute = n.getMinute();
        int second = n.getSecond();
        int nanos = n.getNano();
        System.out.println(hour + ":" + minute + ":" + second + "." +
nanos);//21:55:39.569000000

    }

}
```

增加或减少指定时间

在指定的时间基础之上增加或减少时间 (年、月、周、天、时、分、秒、纳秒)

举例

```java
package com.itlaobing.note;

import java.time.LocalDateTime;

public class LocalDateTimeTest3 {
    public static void main(String[] args) {

        LocalDateTime dt = LocalDateTime.now();
        System.out.println(dt);

        LocalDateTime dt1 = dt.plusYears(-3); // 减少 3 年
        System.out.println(dt1);

        LocalDateTime dt2 = dt.plusMonths(5); // 增加 5 个月
        System.out.println(dt2);

        LocalDateTime dt3 = dt.plusWeeks(10); // 增加 10 周
        System.out.println(dt3);

        LocalDateTime dt4 = dt.plusDays(15); // 增加 15 天
        System.out.println(dt4);

        LocalDateTime dt5 = dt.plusHours(8); // 增加 8 小时
        System.out.println(dt5);

        LocalDateTime dt6 = dt.plusMinutes(30); // 增加 30 分钟
        System.out.println(dt6);

        LocalDateTime dt7 = dt.plusSeconds(50); // 增加 50 秒
        System.out.println(dt7);

        LocalDateTime dt8 = dt.plusNanos(100300500L); // 增加 100300500L 纳秒
        System.out.println(dt8);

    }
}

```

（自己补充`equals`、`isEqual`、`isAfter`、`isBefore`方法的测试吧）

转换

LocalDate转LocalDateTime

LocalDate

- `atStartOfDay()` 将时间初始化为 00:00
- `atTime(int hours, int minutes[, int seconds[, int nanoOfSecond]])` 将时间初始化为指定的值
- `atTime(LocalTime time)` 以指定的时间

LocalDateTime

- `of(LocalDate date, LocalTime time)`

```java
package com.itlaobing.note;

import java.time.LocalDate;
import java.time.LocalDateTime;

public class LocalDateTest4 {

    public static void main(String[] args) {
        LocalDate n = LocalDate.now();
        System.out.println(n);

        // LocalDate  ---> LocalDateTime
        LocalDateTime datetime = n.atStartOfDay();//一天的开始
        System.out.println(datetime);

        LocalDateTime first = n.atTime(10, 20);//在某个时间点
        System.out.println(first);

        LocalDateTime second = n.atTime(10, 20, 30);
        System.out.println(second);

        LocalDateTime third = n.atTime(10, 20, 30, 100200300);
        System.out.println(third);
    }

}

```

LocalTime转LocalDateTime

```java
package com.itlaobing.note;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;

public class LocalTimeTest4 {

    public static void main(String[] args) {
        LocalTime time = LocalTime.now();
        System.out.println(time); //22:03:07.970
        // LocalTime  --->  LocalDateTime
        LocalDate date = LocalDate.of(2019, 10, 24);

        LocalDateTime dateTime = time.atDate(date);// 在某一天
        System.out.println(dateTime);
    }

}

```

LocalDateTime 转 LocalDate 或 LocalTime

```java
package com.itlaobing.note;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;

public class LocalDateTimeTest4 {

    public static void main(String[] args) {
        LocalDateTime datetime = LocalDateTime.now();
        System.out.println(datetime);

        // LocalDateTime ---> LocalDate
        LocalDate date = datetime.toLocalDate();
        System.out.println(date);

        // LocalDateTime ---> LocalTime
        LocalTime time = datetime.toLocalTime();
        System.out.println(time);
    }

}

```

Date转LocalDateTime

```java
package com.itlaobing.note;

import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.util.Date;

public class DateTest2 {

    public static void main(String[] args) {
        /**
         * 将Date对象转换成LocalDate/LocalTime/LocalDateTime
         *    1. 将java.util.Date对象所表示的瞬间转换成java.time.Instant
         *    2. 通过java.time.Instant获取具有时区的日期时间java.time.ZonedDateTime
         *    3. 将java.time.ZonedDateTime转换成相应类型
         */
        final long ms = 1000L * 60 * 60 * 24 * 365 * 26;
        Date date = new Date(ms);
        System.out.println(date);//Tue Dec 26 08:00:00 CST 1995

        System.out.println("===============");
        //Instant指在时间线上的瞬间点
        Instant instant = date.toInstant(); // Date ==> java.time.Instant
        //ZoneId指一个时区ID
        ZoneId zone = ZoneId.of("Asia/Shanghai");//时区设置成亚洲/上海时间
        //ZonedDateTime指具有时区的日期时间
        ZonedDateTime zonedDateTime = instant.atZone(zone);

        LocalDateTime datetime = zonedDateTime.toLocalDateTime();
        System.out.println(datetime);//1995-12-26T08:00

        LocalDate d = zonedDateTime.toLocalDate();
        System.out.println(d);//1995-12-26
        LocalTime t = zonedDateTime.toLocalTime();
        System.out.println(t);//08:00
    }

}

```

LocalDateTime转Date

```java
package com.itlaobing.note;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.util.Date;

public class LocalDateTimeTest5 {

    public static void main(String[] args) {
        /**
         * 将LocalDateTime对象转换成Date对象
         *   1. 获取带时区的java.time.ZonedDateTime对象
         *   2. 将java.time.ZonedDateTime对象转换成表示某一瞬间的java.time.Instant对象
         *   3. 通过Date.from方法，将java.time.Instant对象转换成Date对象
         */
        LocalDateTime datetime = LocalDateTime.of(1999, 11, 22, 6, 30, 7);

        ZoneId zone = ZoneId.systemDefault();//获取系统默认时区

        ZonedDateTime zonedDateTime = ZonedDateTime.of(datetime, zone);

        Instant instant = Instant.from(zonedDateTime);

        Date date = Date.from(instant);

        System.out.println(date);//Mon Nov 22 06:30:07 CST 1999
    }

}

```