

常用API

一、StringBuilder类

- StringBuilder代表可变字符串对象，相当于是一个容器，它里面的字符串是可以改变的，就是用来操作字符串的。
- 好处：StringBuilder比String更合适做字符串的修改操作，效率更高，代码也更加简洁。

1.1 StringBuilder方法演示

接下来我们用代码演示一下StringBuilder的用法

```
public class Test{
    public static void main(String[] args){
        StringBuilder sb = new StringBuilder("itehima");

        //1.拼接内容
        sb.append(12);
        sb.append("黑马");
        sb.append(true);

        //2.append方法，支持临时编程
        sb.append(666).append("黑马2").append(666);
        System.out.println(sb); //打印：12黑马666黑马2666

        //3.反转操作
        sb.reverse();
        System.out.println(sb); //打印：6662马黑666马黑21

        //4.返回字符串的长度
        System.out.println(sb.length());

        //5.StringBuilder还可以转换为字符串
        String s = sb.toString();
        System.out.println(s); //打印：6662马黑666马黑21
    }
}
```

为什么要用StringBuilder对字符串进行操作呢？因为它的效率比String更高，我们可以下面两段代码验证一下。

```
public class Test {
    public static void main(String[] args) {
        String s = "";
        for (int i = 0; i < 1000000; i++) {
            s = s + "abc";
        }
        System.out.println(s);
    }
}
```



```
public class Test {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        for(int i = 0; i < 1000000; i++) {
            sb.append("abc");
        }
        System.out.println(sb);
    }
}
```



经过我的验证，直接使用String拼接100万次，等了1分钟，还没结束，我等不下去了；但是使用StringBuilder做拼接，不到1秒钟出结果了。

1.2 StringBuilder应用案例

接下来，我们通过一个案例把StringBuilder运用下，案例需求如下图所示

代码如下

```
public class Test{
    public static void main(String[] args){
        String str = getArrayData( new int[]{11,22,33});
        System.out.println(str);
    }

    //方法作用：将int数组转换为指定格式的字符串
    public static String getArrayData(int[] arr){
        //1.判断数组是否为null
        if(arr==null){
            return null;
        }
        //2.如果数组不为null，再遍历，并拼接数组中的元素
        StringBuilder sb = new StringBuilder("");
        for(int i=0; i<arr.length; i++){
            if(i==arr.length-1){
                sb.append(arr[i]).append("");
            }else{
                sb.append(arr[i]).append(",");
            }
        }
        //3、把StringBuilder转换为String，并返回。
        return sb.toString();
    }
}
```

二、StringJoiner类

接下来，我们学习一个类叫做StringJoiner，学习这个类干嘛用呢？是因为我们前面使用StringBuilder拼接字符串的时，代码写起来还是有一点麻烦，而StringJoiner号称是拼接神器，不仅效率高，而且代码简洁。

下面演示一下StringJoiner的基本使用

```
public class Test{
    public static void main(String[] args){
        StringJoiner s = new StringJoiner(",");
        s.add("java1");
        s.add("java2");
        s.add("java3");
        System.out.println(s); //结果为: java1,java2,java3

        //参数1: 间隔符
        //参数2: 开头
        //参数3: 结尾
        StringJoiner s1 = new StringJoiner(",","[";"]");
    }
}
```

```

        s1.add("java1");
        s1.add("java2");
        s1.add("java3");
        System.out.println(s1); //结果为: [java1,java2,java3]
    }
}

```

使用StringJoiner改写前面把数组转换为字符串的案例

```

public class Test{
    public static void main(String[] args){
        String str = getArrayData( new int[]{11,22,33});
        System.out.println(str);
    }

    //方法作用：将int数组转换为指定格式的字符串
    public static String getArrayData(int[] arr){
        //1.判断数组是否为null
        if(arr==null){
            return null;
        }
        //2.如果数组不为null，再遍历，并拼接数组中的元素
        StringJoiner s = new StringJoiner(", ","[",""]");
        for(int i=0; i<arr.length; i++){
            //加""是因为add方法的参数要的是String类型
            s.add(String.valueOf(arr[i]));
        }
        //3、把StringJoiner转换为String，并返回。
        return s.toString();
    }
}

```

三、Math类

Math是数学的意思，该类提供了很多个进行数学运算的方法，如求绝对值，求最大值，四舍五入等，话不多说，直接上代码。

```

public class MathTest {
    public static void main(String[] args) {
        // 目标：了解下Math类提供的常见方法。
        // 1、public static int abs(int a): 取绝对值（拿到的结果一定是正数）
        //     public static double abs(double a)
        System.out.println(Math.abs(-12)); // 12
        System.out.println(Math.abs(123)); // 123
        System.out.println(Math.abs(-3.14)); // 3.14

        // 2、public static double ceil(double a): 向上取整
        System.out.println(Math.ceil(4.0000001)); // 5.0
        System.out.println(Math.ceil(4.0)); // 4.0

        // 3、public static double floor(double a): 向下取整
        System.out.println(Math.floor(4.9999999)); // 4.0
        System.out.println(Math.floor(4.0)); // 4.0
    }
}

```

```

// 4、public static long round(double a): 四舍五入
System.out.println(Math.round(3.4999)); // 3
System.out.println(Math.round(3.50001)); // 4

// 5、public static int max(int a, int b): 取较大值
//   public static int min(int a, int b): 取较小值
System.out.println(Math.max(10, 20)); // 20
System.out.println(Math.min(10, 20)); // 10

// 6、public static double pow(double a, double b): 取次方
System.out.println(Math.pow(2, 3)); // 2的3次方   8.0
System.out.println(Math.pow(3, 2)); // 3的2次方   9.0

// 7、public static double random(): 取随机数 [0.0 , 1.0) (包前不包后)
System.out.println(Math.random());
}
}

```

四、System类

接下来，学习一个System类，这是系统类，提供了一些获取获取系统数据的方法。比如获取系统时间。

```

/**
 * 目标：了解下System类的常见方法。
 */
public class SystemTest {
    public static void main(String[] args) {

        // 1、public static void exit(int status):
        //   终止当前运行的Java虚拟机。
        //   该参数用作状态代码；按照惯例，非零状态代码表示异常终止。
        System.exit(0); // 人为的终止虚拟机。（不要使用）

        // 2、public static long currentTimeMillis():
        //   获取当前系统的时间
        //   返回的是long类型的时间毫秒值：指的是从1970-1-1 0:0:0开始走到此刻的总的毫秒
        //   值，1s = 1000ms
        long time = System.currentTimeMillis();
        System.out.println(time);

        for (int i = 0; i < 1000000; i++) {
            System.out.println("输出了: " + i);
        }

        long time2 = System.currentTimeMillis();
        System.out.println((time2 - time) / 1000.0 + "s");
    }
}

```

五、Runtime类

接下来，我们再学习一个Java的运行类，叫Runtime类。这个类可以用来获取JVM的一些信息，也可以用这个类去执行其他的程序。话不多说，上代码。

```
/**
 * 目标：了解下Runtime的几个常见方法。
 */
public class RuntimeTest {
    public static void main(String[] args) throws IOException,
        InterruptedException {

        // 1、public static Runtime getRuntime() 返回与当前Java应用程序关联的运行时对
        象。
        Runtime r = Runtime.getRuntime();

        // 2、public void exit(int status) 终止当前运行的虚拟机,该参数用作状态代码；按
        照惯例，非零状态代码表示异常终止。
        // r.exit(0);

        // 3、public int availableProcessors()：获取虚拟机能够使用的处理器数。
        System.out.println(r.availableProcessors());

        // 4、public long totalMemory() 返回Java虚拟机中的内存总量。
        System.out.println(r.totalMemory()/1024.0/1024.0 + "MB"); // 1024 = 1k
        1024 * 1024 = 1M

        // 5、public long freeMemory() 返回Java虚拟机中的可用内存量
        System.out.println(r.freeMemory()/1024.0/1024.0 + "MB");

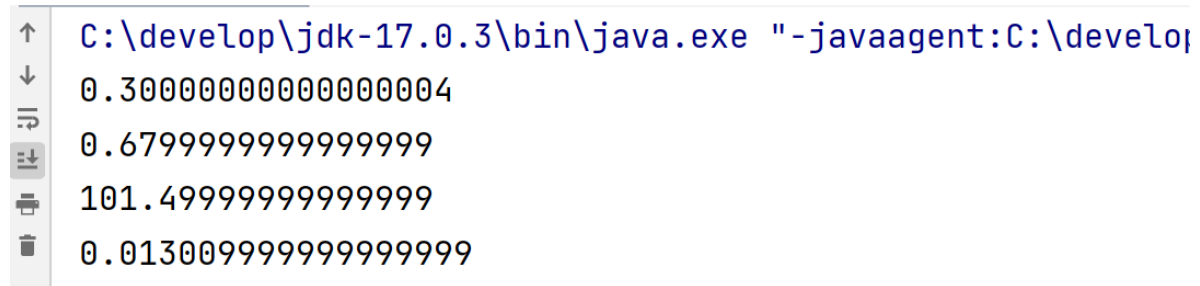
        // 6、public Process exec(String command) 启动某个程序，并返回代表该程序的
        对
        象。
        // r.exec("D:\\soft\\XMind\\XMind.exe");
        Process p = r.exec("QQ");
        Thread.sleep(5000); // 让程序在这里暂停5s后继续往下走！！
        p.destroy(); // 销毁！关闭程序！
    }
}
```

六、BigDecimal类

各位同学，接下来我们学习的这个类叫BigDecimal，至于它是干什么用的，我们先不说。我们先看一段代码，看这个代码有什么问题？再说BigDeimal这个类是干什么用的，这样会更好理解一些。

```
public class Test {
    public static void main(String[] args) {
        System.out.println(0.1 + 0.2);
        System.out.println(1.0 - 0.32);
        System.out.println(1.015 * 100);
        System.out.println(1.301 / 100);
    }
}
```

运行以上代码，我们会发现，结果并和我们想看到的不太一样。如下图所示

A terminal window with a dark background and light blue text. The command 'C:\develop\jdk-17.0.3\bin\java.exe "-javaagent:C:\develop\...' is entered. The output shows five lines of decimal numbers: 0.30000000000000000004, 0.6799999999999999, 101.49999999999999, and 0.013009999999999999.

```
C:\develop\jdk-17.0.3\bin\java.exe "-javaagent:C:\develop\...
0.30000000000000000004
0.6799999999999999
101.49999999999999
0.013009999999999999
```

为了解决计算精度损失的问题，Java给我们提供了BigDecimal类，它提供了一些方法可以对数据进行四则运算，而且不丢失精度，同时还可以保留指定的小数位。下面看代码，演示一下

```
public class Test2 {
    public static void main(String[] args) {
        // 目标：掌握BigDecimal进行精确运算的方案。
        double a = 0.1;
        double b = 0.2;

        // 1、把浮点型数据封装成BigDecimal对象，再来参与运算。
        // a、public BigDecimal(double val) 得到的BigDecimal对象是无法精确计算浮点型数据的。 注意：不推荐使用这个，
        // b、public BigDecimal(String val) 得到的BigDecimal对象是可以精确计算浮点型数据的。 可以使用。
        // c、public static BigDecimal valueOf(double val)：通过这个静态方法得到的BigDecimal对象是可以精确运算的。是最好的方案。
        BigDecimal a1 = BigDecimal.valueOf(a);
        BigDecimal b1 = BigDecimal.valueOf(b);

        // 2、public BigDecimal add(BigDecimal augend)：加法
        BigDecimal c1 = a1.add(b1);
        System.out.println(c1);

        // 3、public BigDecimal subtract(BigDecimal augend)：减法
        BigDecimal c2 = a1.subtract(b1);
        System.out.println(c2);

        // 4、public BigDecimal multiply(BigDecimal augend)：乘法
        BigDecimal c3 = a1.multiply(b1);
        System.out.println(c3);

        // 5、public BigDecimal divide(BigDecimal b)：除法
        BigDecimal c4 = a1.divide(b1);
        System.out.println(c4);

        //
        BigDecimal d1 = BigDecimal.valueOf(0.1);
        //
        BigDecimal d2 = BigDecimal.valueOf(0.3);
        //
        BigDecimal d3 = d1.divide(d2);
        //
        System.out.println(d3);

        // 6、public BigDecimal divide(另一个BigDecimal对象，精确几位，舍入模式)：除法，可以设置精确几位。
        BigDecimal d1 = BigDecimal.valueOf(0.1);
        BigDecimal d2 = BigDecimal.valueOf(0.3);
        BigDecimal d3 = d1.divide(d2, 2, RoundingMode.HALF_UP); // 0.33
        System.out.println(d3);
    }
}
```

```

        // 7、public double doubleValue() : 把BigDecimal对象又转换成double类型的数据。
        //print(d3);
        //print(c1);
        double db1 = d3.doubleValue();
        double db2 = c1.doubleValue();
        print(db1);
        print(db2);
    }

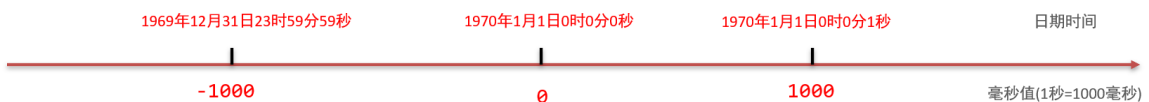
    public static void print(double a){
        System.out.println(a);
    }
}

```

五、Date类

接下来，我们学习一下Date类，Java中是由这个类的对象用来表示日期或者时间。

Date对象记录的时间是用毫秒值来表示的。Java语言规定，1970年1月1日0时0分0秒认为是时间的起点，此时记作0，那么1000（1秒=1000毫秒）就表示1970年1月1日0时0分1秒，依次内推。



下面是Date类的构造方法，和常见的成员方法，利用这些API写代码尝试一下

构造器	说明
public Date()	创建一个Date对象，代表的是系统当前此刻日期时间。
public Date(long time)	把时间毫秒值转换成Date日期对象。

常见方法	说明
public long getTime()	返回从1970年1月1日 00:00:00走到此刻的总的毫秒数
public void setTime(long time)	设置日期对象的时间为当前时间毫秒值对应的时间

```

public class Test1Date {
    public static void main(String[] args) {
        // 目标：掌握Date日期类的使用。
        // 1、创建一个Date的对象：代表系统当前时间信息的。
        Date d = new Date();
        System.out.println(d);

        // 2、拿到时间毫秒值。
        long time = d.getTime();
        System.out.println(time);
    }
}

```

```

// 3、把时间毫秒值转换成日期对象： 2s之后的时间是多少。
time += 2 * 1000;
Date d2 = new Date(time);
System.out.println(d2);

// 4、直接把日期对象的时间通过setTime方法进行修改
Date d3 = new Date();
d3.setTime(time);
System.out.println(d3);
}
}

```

六、SimpleDateFormat类

各位同学，前面我们打印Date对象时，发现打印输出的日期格式我们并不喜欢，是不是？你们喜欢那种格式呢？是不是像下面页面中这种格式啊？接下来我们学习的SimpleDateFormat类就可以转换Date对象表示日期时间的显示格式。

- 我们把Date对象转换为指定格式的日期字符串这个操作，叫做**日期格式化**，
- 反过来把指定格式的日期字符串转换为Date对象的操作，叫做**日期解析**。



接下来，我们先演示一下日期格式化，需要用到如下的几个方法

常见构造器	说明
public SimpleDateFormat(String pattern)	创建简单日期格式化对象，并封装时间的格式

格式化时间的方法	说明
public final String format(Date date)	将日期格式化成日期/时间字符串
public final String format(Object time)	将时间毫秒值式化成日期/时间字符串

注意：创建SimpleDateFormat对象时，在构造方法的参数位置传递日期格式，而日期格式是由一些特定的字母拼接而来的。我们需要记住常用的几种日期/时间格式

字母	表示含义
yyyy	年
MM	月
dd	日
HH	时
mm	分
ss	秒
SSS	毫秒

"2022年12月12日" 的格式是 "yyyy年MM月dd日"

"2022-12-12 12:12:12" 的格式是 "yyyy-MM-dd HH:mm:ss"

按照上面的格式可以任意拼接，但是字母不能写错

最后，上代码演示一下

```
public class Test2SimpleDateFormat {
    public static void main(String[] args) throws ParseException {
        // 目标：掌握SimpleDateFormat的使用。
        // 1、准备一些时间
        Date d = new Date();
        System.out.println(d);

        long time = d.getTime();
        System.out.println(time);

        // 2、格式化日期对象，和时间 毫秒值。
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss EEE
a");

        String rs = sdf.format(d);
        String rs2 = sdf.format(time);
        System.out.println(rs);
        System.out.println(rs2);
        System.out.println("-----");

        // 目标：掌握SimpleDateFormat解析字符串时间 成为日期对象。
        String dateStr = "2022-12-12 12:12:11";
        // 1、创建简单日期格式化对象，指定的时间格式必须与被解析的时间格式一模一样，否则程序
        会出bug.
        SimpleDateFormat sdf2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Date d2 = sdf2.parse(dateStr);
        System.out.println(d2);
    }
}
```

日期格式化&解析案例



需求

- 小贾下单并付款的时间为：2023年11月11日 0:01:18
- 小皮下单并付款的时间为：2023年11月11日 0:10:51
- 请用代码说明这两位同学有没有参加上秒杀活动？

```
public class Test3 {  
    public static void main(String[] args) throws ParseException {  
        // 目标：完成秒杀案例。  
        // 1、把开始时间、结束时间、小贾下单时间、小皮下单时间拿到程序中来。  
        String start = "2023年11月11日 0:0:0";  
        String end = "2023年11月11日 0:10:0";  
        String xj = "2023年11月11日 0:01:18";  
        String xp = "2023年11月11日 0:10:57";  
  
        // 2、把字符串的时间解析成日期对象。  
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss");  
        Date startDt = sdf.parse(start);  
        Date endDt = sdf.parse(end);  
        Date xjDt = sdf.parse(xj);  
        Date xpDt = sdf.parse(xp);  
  
        // 3、开始判断小皮和小贾是否秒杀成功了。  
        // 把日期对象转换成时间毫秒值来判断  
        long startTime = startDt.getTime();  
        long endTime = endDt.getTime();  
        long xjTime = xjDt.getTime();  
        long xpTime = xpDt.getTime();  
  
        if(xjTime >= startTime && xjTime <= endTime){  
            System.out.println("小贾您秒杀成功了~~");  
        }else {  
            System.out.println("小贾您秒杀失败了~~");  
        }  
  
        if(xpTime >= startTime && xpTime <= endTime){  
            System.out.println("小皮您秒杀成功了~~");  
        }else {  
            System.out.println("小皮您秒杀失败了~~");  
        }  
    }  
}
```

七、Calendar类

学完Date类和SimpleDateFormat类之后，我们再学习一个和日期相关的类，它是Calendar类。Calendar类表示日历，它提供了一些比Date类更好用的方法。

比如下面的案例，用Date类就不太好做，而用Calendar就特别方便。因为Calendar类提供了方法可以直接对日历中的年、月、日、时、分、秒等进行运算。

需求：将2023年09月10日 增加一个月



Calendar

- 代表的是系统此刻时间对应的日历。
- 通过它可以单独获取、修改时间中的年、月、日、时、分、秒等。

Calendar日历类的常见方法

方法名	说明
<code>public static Calendar getInstance()</code>	获取当前日历对象
<code>public int get(int field)</code>	获取日历中的某个信息。
<code>public final Date getTime()</code>	获取日期对象。
<code>public long getTimeInMillis()</code>	获取时间毫秒值
<code>public void set(int field,int value)</code>	修改日历的某个信息。
<code>public void add(int field,int amount)</code>	为某个信息增加/减少指定的值

注意：calendar是可变对象，一旦修改后其对象本身表示的时间将产生变化。

```
public class Test4Calendar {
    public static void main(String[] args) {
        // 目标：掌握Calendar的使用和特点。
        // 1、得到系统此刻时间对应的日历对象。
        Calendar now = Calendar.getInstance();
        System.out.println(now);

        // 2、获取日历中的某个信息
        int year = now.get(Calendar.YEAR);
        System.out.println(year);

        int days = now.get(Calendar.DAY_OF_YEAR);
        System.out.println(days);

        // 3、拿到日历中记录的日期对象。
        Date d = now.getTime();
        System.out.println(d);

        // 4、拿到时间毫秒值
        long time = now.getTimeInMillis();
        System.out.println(time);

        // 5、修改日历中的某个信息
        now.set(Calendar.MONTH, 9); // 修改月份成为10月份。
    }
}
```

```

        now.set(Calendar.DAY_OF_YEAR, 125); // 修改成一年中的第125天。
        System.out.println(now);

        // 6、为某个信息增加或者减少多少
        now.add(Calendar.DAY_OF_YEAR, 100);
        now.add(Calendar.DAY_OF_YEAR, -10);
        now.add(Calendar.DAY_OF_MONTH, 6);
        now.add(Calendar.HOUR, 12);
        now.set(2026, 11, 22);
        System.out.println(now);
    }
}

```

八、为什么JDK8要新增日期类

```

/**
 * 目标：搞清楚为什么要用JDK 8开始新增的时间类。
 */
public class Test {
    public static void main(String[] args) {
        // 传统的时间类（Date、SimpleDateFormat、Calendar）存在如下问题：
        // 1、设计不合理，使用不方便，很多都被淘汰了。
        Date d = new Date();
        //System.out.println(d.getYear() + 1900);

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

        // 2、都是可变对象，修改后会丢失最开始的时间信息。

        // 3、线程不安全。

        // 4、不能精确到纳秒，只能精确到毫秒。
        // 1秒 = 1000毫秒
        // 1毫秒 = 1000微妙
        // 1微妙 = 1000纳秒
    }
}

```

九、JDK8日期、时间、日期时间

接下来，我们学习一下JDK8新增的日期类。为什么以前的Date类就可以表示日期，为什么要有新增的日期类呢？原因如下

JDK8之前 传统的时间API

- 1、设计不合理，使用不方便，很多都被淘汰了。
- 2、都是可变对象，修改后会丢失最开始的时间信息。
- 3、线程不安全。
- 4、只能精确到毫秒。

JDK8开始之后 新增的时间API

- 1、设计更合理，功能丰富，使用更方便。
- 2、都是不可变对象，修改后会返回新的时间对象，不会丢失最开始的时间。
- 3、线程安全。
- 4、能精确到毫秒、纳秒。

Tip:

1秒=1000毫秒;

1毫秒=1000微秒;

1微秒=1000纳秒;

不推荐！

推荐使用！

JDK8新增的日期类分得更细致一些，比如表示年月日用LocalDate类、表示时间秒用LocalTime类、而表示年月日时分秒用LocalDateTime类等；除了这些类还提供了对时区、时间间隔进行操作的类等。它们几乎把对日期/时间的所有操作都通过了API方法，用起来特别方便。

01
STEP

代替Calendar

LocalDate: 年、月、日
LocalTime: 时、分、秒
LocalDateTime: 年、月、日 时、分、秒
ZoneId: 时区
ZonedDateTime: 带时区的时间

02
STEP

代替Date

Instant: 时间戳/时间线

03
STEP

代替SimpleDateFormat

DateTimeFormatter: 用于时间的格式化和解析

04
STEP

其他补充

Period: 时间间隔 (年, 月, 日)

Duration: 时间间隔 (时、分、秒, 纳秒)

先学习表示日期、时间、日期时间的类；有LocalDate、LocalTime、以及LocalDateTime类。仔细阅读代码，你会发现这三个类的用法套路都是一样的。

- LocalDate类的基本使用

```
public class Test1_LocalDate {  
    public static void main(String[] args) {  
        // 0、获取本地日期对象(不可变对象)  
        LocalDate ld = LocalDate.now(); // 年 月 日  
        System.out.println(ld);  
  
        // 1、获取日期对象中的信息  
        int year = ld.getYear(); // 年  
        int month = ld.getMonthValue(); // 月(1-12)  
        int day = ld.getDayOfMonth(); // 日  
        int dayOfYear = ld.getDayOfYear(); // 一年中的第几天  
        int dayOfWeek = ld.getDayOfWeek().getValue(); // 星期几  
        System.out.println(year);  
        System.out.println(day);  
        System.out.println(dayOfWeek);  
  
        // 2、直接修改某个信息: withYear、withMonth、withDayOfMonth、withDayOfYear  
        LocalDate ld2 = ld.withYear(2099);  
    }  
}
```

```

        LocalDate ld3 = ld.withMonth(12);
        System.out.println(ld2);
        System.out.println(ld3);
        System.out.println(ld);

        // 3、把某个信息加多少: plusYears、plusMonths、plusDays、plusWeeks
        LocalDate ld4 = ld.plusYears(2);
        LocalDate ld5 = ld.plusMonths(2);

        // 4、把某个信息减多少: minusYears、minusMonths、minusDays、minusWeeks
        LocalDate ld6 = ld.minusYears(2);
        LocalDate ld7 = ld.minusMonths(2);

        // 5、获取指定日期的LocalDate对象: public static LocalDate of(int year, int
month, int dayOfMonth)
        LocalDate ld8 = LocalDate.of(2099, 12, 12);
        LocalDate ld9 = LocalDate.of(2099, 12, 12);

        // 6、判断2个日期对象, 是否相等, 在前还是在后: equals isBefore isAfter
        System.out.println(ld8.equals(ld9)); // true
        System.out.println(ld8.isAfter(ld)); // true
        System.out.println(ld8.isBefore(ld)); // false
    }
}

```

- LocalTime类的基本使用

```

public class Test2_LocalTime {
    public static void main(String[] args) {
        // 0、获取本地时间对象
        LocalTime lt = LocalTime.now(); // 时 分 秒 纳秒 不可变的
        System.out.println(lt);

        // 1、获取时间中的信息
        int hour = lt.getHour(); //时
        int minute = lt.getMinute(); //分
        int second = lt.getSecond(); //秒
        int nano = lt.getNano(); //纳秒

        // 2、修改时间: withHour、withMinute、withSecond、withNano
        LocalTime lt3 = lt.withHour(10);
        LocalTime lt4 = lt.withMinute(10);
        LocalTime lt5 = lt.withSecond(10);
        LocalTime lt6 = lt.withNano(10);

        // 3、加多少: plusHours、plusMinutes、plusSeconds、plusNanos
        LocalTime lt7 = lt.plusHours(10);
        LocalTime lt8 = lt.plusMinutes(10);
        LocalTime lt9 = lt.plusSeconds(10);
        LocalTime lt10 = lt.plusNanos(10);

        // 4、减多少: minusHours、minusMinutes、minusSeconds、minusNanos
        LocalTime lt11 = lt.minusHours(10);
        LocalTime lt12 = lt.minusMinutes(10);
        LocalTime lt13 = lt.minusSeconds(10);
        LocalTime lt14 = lt.minusNanos(10);
    }
}

```

```

// 5、获取指定时间的LocalTime对象：
// public static LocalTime of(int hour, int minute, int second)
LocalTime lt15 = LocalTime.of(12, 12, 12);
LocalTime lt16 = LocalTime.of(12, 12, 12);

// 6、判断2个时间对象，是否相等，在前还是在后： equals isBefore isAfter
System.out.println(lt15.equals(lt16)); // true
System.out.println(lt15.isAfter(lt)); // false
System.out.println(lt15.isBefore(lt)); // true

    }
}

```

- LocalDateTime类的基本使用

```

public class Test3_LocalDateTime {
    public static void main(String[] args) {
        // 0、获取本地日期和时间对象。
        LocalDateTime ldt = LocalDateTime.now(); // 年 月 日 时 分 秒 纳秒
        System.out.println(ldt);

        // 1、可以获取日期和时间的全部信息
        int year = ldt.getYear(); // 年
        int month = ldt.getMonthValue(); // 月
        int day = ldt.getDayOfMonth(); // 日
        int dayOfYear = ldt.getDayOfYear(); // 一年中的第几天
        int dayOfWeek = ldt.getDayOfWeek().getValue(); // 获取是周几
        int hour = ldt.getHour(); // 时
        int minute = ldt.getMinute(); // 分
        int second = ldt.getSecond(); // 秒
        int nano = ldt.getNano(); // 纳秒

        // 2、修改时间信息：
        // withYear withMonth withDayOfMonth withDayOfYear withHour
        // withMinute withSecond withNano
        LocalDateTime ldt2 = ldt.withYear(2029);
        LocalDateTime ldt3 = ldt.withMinute(59);

        // 3、加多少：
        // plusYears plusMonths plusDays plusWeeks plusHours plusMinutes
        // plusSeconds plusNanos
        LocalDateTime ldt4 = ldt.plusYears(2);
        LocalDateTime ldt5 = ldt.plusMinutes(3);

        // 4、减多少：
        // minusDays minusYears minusMonths minusWeeks minusHours minusMinutes
        // minusSeconds minusNanos
        LocalDateTime ldt6 = ldt.minusYears(2);
        LocalDateTime ldt7 = ldt.minusMinutes(3);

        // 5、获取指定日期和时间的LocalDateTime对象：
        // public static LocalDateTime of(int year, Month month, int dayOfMonth,
        // int hour,
        // int minute, int second, int
        // nanoOfSecond)
        LocalDateTime ldt8 = LocalDateTime.of(2029, 12, 12, 12, 12, 12, 1222);
    }
}

```

```

        LocalDateTime ldt9 = LocalDateTime.of(2029, 12, 12, 12, 12, 1222);

        // 6、判断2个日期、时间对象，是否相等，在前还是在后： equals、isBefore、isAfter
        System.out.println(ldt9.equals(ldt8));
        System.out.println(ldt9.isAfter(ldt));
        System.out.println(ldt9.isBefore(ldt));

        // 7、可以把LocalDateTime转换成LocalDate和LocalTime
        // public LocalDate toLocalDate()
        // public LocalTime toLocalTime()
        // public static LocalDateTime of(LocalDate date, LocalTime time)
        LocalDate ld = ldt.toLocalDate();
        LocalTime lt = ldt.toLocalTime();
        LocalDateTime ldt10 = LocalDateTime.of(ld, lt);

    }
}

```

十、JDK8日期（时区）

接着，我们学习代表时区的两个类。由于世界各国与地区的经度不同，各地区的时间也有所不同，因此会划分为不同的时区。每一个时区的时间也不太一样。



代替 Calendar

LocalDate: 年、月、日

LocalTime: 时、分、秒

LocalDateTime: 年、月、日 时、分、秒

ZoneId: 时区

ZonedDateTime: 带时区的时间

美国时间

```

public class Test4_ZoneId_ZonedDateTime {
    public static void main(String[] args) {
        // 目标：了解时区和带时区的时间。
        // 1、ZoneId的常见方法：
        // public static ZoneId systemDefault(): 获取系统默认的时区
        ZoneId zoneId = ZoneId.systemDefault();
        System.out.println(zoneId.getId());
        System.out.println(zoneId);

        // public static Set<String> getAvailableZoneIds(): 获取Java支持的全部时区
        System.out.println(ZoneId.getAvailableZoneIds());

        // public static ZoneId of(String zoneId) : 把某个时区id封装成ZoneId对象。
        ZoneId zoneId1 = ZoneId.of("America/New_York");

        // 2、ZonedDateTime: 带时区的时间。
        // public static ZonedDateTime now(ZoneId zone): 获取某个时区的
        ZonedDateTime对象。
    }
}

```



```

        ZonedDateTime now = ZonedDateTime.now(zoneId1);
        System.out.println(now);

        // 世界标准时间了
        ZonedDateTime now1 = ZonedDateTime.now(Clock.systemUTC());
        System.out.println(now1);

        // public static ZonedDateTime now(): 获取系统默认时区的ZonedDateTime对象
        ZonedDateTime now2 = ZonedDateTime.now();
        System.out.println(now2);

        // Calendar instance =
        Calendar.getInstance(TimeZone.getTimeZone(zoneId1));
    }
}

```

十一、JDK8日期 (Instant类)

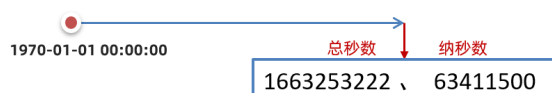
接下来，我们来学习Instant这个类。通过获取Instant的对象可以拿到此刻的时间，该时间由两部分组成：从1970-01-01 00:00:00 开始走到此刻的总秒数+不够1秒的纳秒数。



代替Date

Instant：时间线上的某个时刻/时间戳

该类提供的方法如下图所示，可以用来获取当前时间，也可以对时间进行加、减、获取等操作。



方法名	说明
public static Instant now()	获取当前时间的Instant对象 (标准时间)
public long getEpochSecond()	获取从1970-01-01T00:00:00开始记录的秒数。
public int getNano()	从时间线开始，获取从第二个开始的纳秒数
plusMillis plusSeconds plusNanos	增加时间系列的方法
minusMillis minusSeconds minusNanos	减少时间系列的方法
equals, isBefore, isAfter	判断时间系列的方法

Tip:
 1秒=1000毫秒;
 1毫秒=1000微秒;
 1微秒=1000纳秒;
 1秒 = 1000 000 000纳秒

作用：可以用来记录代码的执行时间，或用于记录用户操作某个事件的时间点。

```

/**
 * 目标：掌握Instant的使用。
 */
public class Test5_Instant {
    public static void main(String[] args) {
        // 1、创建Instant的对象，获取此刻时间信息
        Instant now = Instant.now(); // 不可变对象
    }
}

```

```

// 2、获取总秒数
long second = now.getEpochSecond();
System.out.println(second);

// 3、不够1秒的纳秒数
int nano = now.getNano();
System.out.println(nano);

System.out.println(now);

Instant instant = now.plusNanos(111);

// Instant对象的作用：做代码的性能分析，或者记录用户的操作时间点
Instant now1 = Instant.now();
// 代码执行。。。
Instant now2 = Instant.now();

LocalDateTime l = LocalDateTime.now();
}
}

```

十二、JDK8日期（格式化器）

接下来，我们学习一个新增的日期格式化类，叫DateTimeFormatter。它可以对日期进行格式化和解析。它代替了原来的SimpleDateFormat类。



代替SimpleDateFormat 线程不安全

DateTimeFormatter：格式化器，用于时间的格式化、解析

线程安全

需要用到的方法，如下图所示

DateTimeFormatter

方法名	说明
public static DateTimeFormatter ofPattern(时间格式)	获取格式化器对象
public String format(时间对象)	格式化时间

LocalDateTime提供的格式化、解析时间的方法

方法名	说明
public String format(DateTimeFormatter formatter)	格式化时间
public static LocalDateTime parse(CharSequence text, DateTimeFormatter formatter)	解析时间

接下来，将上面的方法用代码来演示一下

```

/**
 * 目标：掌握JDK 8新增的DateTimeFormatter格式化器的用法。
 */
public class Test6_DateTimeFormatter {
    public static void main(String[] args) {
        // 1、创建一个日期时间格式化器对象出来。
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy年MM月dd日HH:mm:ss");

        // 2、对时间进行格式化
        LocalDateTime now = LocalDateTime.now();
        System.out.println(now);

        String rs = formatter.format(now); // 正向格式化
        System.out.println(rs);

        // 3、格式化时间，其实还有一种方案。
        String rs2 = now.format(formatter); // 反向格式化
        System.out.println(rs2);

        // 4、解析时间：解析时间一般使用LocalDateTime提供的解析方法来解析。
        String dateStr = "2029年12月12日 12:12:11";
        LocalDateTime ldt = LocalDateTime.parse(dateStr, formatter);
        System.out.println(ldt);
    }
}

```

十三、JDK8日期（Period类）

除了上新增的类，JDK8还补充了两个类，一个叫Period类、一个叫Duration类；这两个类可以用来对计算两个时间点的时间间隔。

其中Period用来计算日期间隔（年、月、日），Duration用来计算时间间隔（时、分、秒、纳秒）



其他补充

Period:计算日期间隔（年，月，日）

Duration:计算时间间隔（时、分、秒，纳秒）

先来演示Period类的用法，它的方法如下图所示。可以用来计算两个日期之间相隔的年、相隔的月、相隔的日。**只能两个计算LocalDate对象之间的间隔**

方法名	说明
public static Period between(LocalDate start, LocalDate end)	传入2个日期对象，得到Period对象
public int getYears()	计算隔几年，并返回
public int getMonths()	计算隔几个月，年返回
public int getDays()	计算隔多少天，并返回

```

/**
 * 目标：掌握Period的作用：计算机两个日期相差的年数，月数、天数。
 */
public class Test7_Period {
    public static void main(String[] args) {
        LocalDate start = LocalDate.of(2029, 8, 10);
        LocalDate end = LocalDate.of(2029, 12, 15);

        // 1、创建Period对象，封装两个日期对象。
        Period period = Period.between(start, end);

        // 2、通过period对象获取两个日期对象相差的信息。
        System.out.println(period.getYears());
        System.out.println(period.getMonths());
        System.out.println(period.getDays());
    }
}

```

十四、JDK8日期（Duration类）

接下来，我们学习Duration类。它是用来表示两个时间对象的时间间隔。可以用于计算两个时间对象相差的天数、小时数、分数、秒数、纳秒数；支持LocalTime、LocalDateTime、Instant等时间

方法名	说明
public static Duration between(开始时间对象1,截止时间对象2)	传入2个时间对象，得到Duration对象
public long toDays()	计算隔多少天，并返回
public long toHours()	计算隔多少小时，并返回
public long toMinutes()	计算隔多少分，并返回
public long toSeconds()	计算隔多少秒，并返回
public long toMillis()	计算隔多少毫秒，并返回
public long toNanos()	计算隔多少纳秒，并返回

```

public class Test8_Duration {
    public static void main(String[] args) {
        LocalDateTime start = LocalDateTime.of(2025, 11, 11, 11, 10, 10);
        LocalDateTime end = LocalDateTime.of(2025, 11, 11, 11, 11, 11);
        // 1、得到Duration对象
        Duration duration = Duration.between(start, end);

        // 2、获取两个时间对象间隔的信息
    }
}

```

```
System.out.println(duration.toDays());// 间隔多少天
System.out.println(duration.toHours());// 间隔多少小时
System.out.println(duration.toMinutes());// 间隔多少分
System.out.println(duration.toSeconds());// 间隔多少秒
System.out.println(duration.toMillis());// 间隔多少毫秒
System.out.println(duration.toNanos());// 间隔多少纳秒

    }
}
```