一、处理时间

1.0 Date概念

Date类代表当前所在系统的日期时间信息

Date的构造器

Date的构造器

名称	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。

构造器	说明
public Date(long time)	把时间毫秒值转换成Date日期对象。

Date的常用方法

Date的常用方法

名称	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

Date方法	说明
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

```
    /**
```

```
     * ClassName: DateDemo
     * Description:从现在开始向前走了一小时两分钟1秒
     * date:2022/3/10
     *
     * @author fgcy
     * @since JDK 1.8
     */
    @Test
    public void test1() {
        final Date now = new Date();//通过无参构造获得当前日期对象
        System.out.println(now);
        final long nowTime = now.getTime();//将当前的日期对象转成时间毫秒值
        //将当前的时间毫秒值加【一小时两分钟1秒】毫秒，将值注入到之前的日期对象中，此时日期对象的值已经改变
        now.setTime(nowTime + (3600 + 121) * 1000);
        System.out.println(now);
    }

================================================================================
====================
Thu Mar 10 18:00:42 CST 2022
Thu Mar 10 19:02:43 CST 2022
```

小结

日期对象如何创建，如何获取时间毫秒值?

public Date();//获得一个当前时间的日期对象

public long getTime();//获得当前日期对象的时间毫秒值

public Date(long time);//获得一个指定时间毫秒值时间的日期对象

时间毫秒值怎么恢复成日期对象

public Date(long time);//构造器，根据时间毫秒值获得一个日期对象

public void setTime(long time);//通过当前对象设置时间毫秒值，改变日期对象

2.0 SimpleDateFormat

构造器

构造器

构造器	说明
public SimpleDateFormat(String pattern)	构造一个SimpleDateFormat，使用指定的格式

格式化方法

格式化方法

格式化方法	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

解析方法

解析方法	说明
public Date **parse(String source)**	从给定字符串的开始解析文本以生成日期

格式化字符


```
    @Test
    public void test2() throws Exception{
        final SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日 HH点mm分ss秒 EEE a");
        final Date date = sdf.parse("2021年08月06日 11点11分11秒 星期五 上午");
        System.out.println(sdf.format(date));
        final long targetTime = date.getTime() + (3600L * 24 * 2 + 14 * 3600 + 49 * 60 + 6) * 1000;
        date.setTime(targetTime);
        System.out.println(sdf.format(date));
    }
    ============================================================

2021年08月06日 11点11分11秒 星期五 上午
2021年08月09日 02点00分17秒 星期一 上午
```

案例 秒杀活动

```
@Test
    public void test3() throws Exception {
        String begin = "2020年11月11日 0:00:00";
        String end = "2020年11月11日 0:10:00";
        String jia = "2020年11月11日 0:03:47";
        String pi = "2020年11月11日 0:10:11";

        final SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日
H:mm:ss");
        final Date begins = sdf.parse(begin);
        final Date ends = sdf.parse(end);
        final Date jias = sdf.parse(jia);
        final Date pis = sdf.parse(pi);
        if (pis.after(begins) && pis.before(ends)) {
            System.out.println("小皮秒杀成功");
        } else {
            System.out.println("小皮秒杀失败");
        }

        if (jias.after(begins) && jias.before(ends)) {
            System.out.println("小甲秒杀成功");
        } else {

            System.out.println("小甲秒杀失败");
        }
    }
========================================================================
小皮秒杀失败
小甲秒杀成功
```

二、Calendar

概述

Calendar代表了系统此刻日期对应的日历对象。日期类有年月日时分秒星期等；日历的范围更广；

Calendar是一个抽象类，不能直接创建对象。

常用方法

Calendar日历类创建日历对象的方法：

方法名	说明
public static Calendar getInstance()	获取当前日历对象

Calendar常用方法

方法名	说明
public int get(int field)	取日期中的某个字段信息。
public void set(int field,int value)	修改日历的某个字段信息。
public void add(int field,int amount)	为某个字段增加/减少指定的值
public final Date getTime()	拿到此刻日期对象。
public long getTimeInMillis()	拿到此刻时间毫秒值

```
    @Test
    public void test1() {
        final Calendar cal = Calendar.getInstance();//该类是抽象类，getInstance是
通过调用该类的子类来获取实例
        System.out.println(cal);
        System.out.println(cal.get(Calendar.YEAR) + "年");
        System.out.println(cal.get(Calendar.MONTH) + 1 + "月");
        System.out.println("今年第" + cal.get(Calendar.WEEK_OF_YEAR) + "周");
        System.out.println("本月第" + cal.get(Calendar.WEEK_OF_MONTH) + "周");
        System.out.println(cal.get(Calendar.DAY_OF_MONTH) + "日");
        System.out.println("今年第" + cal.get(Calendar.DAY_OF_YEAR) + "天");
        System.out.println("本周第" + cal.get(Calendar.DAY_OF_WEEK) + "天");
        System.out.println(cal.get(Calendar.HOUR_OF_DAY) + "点");
        System.out.println(cal.get(Calendar.MINUTE) + "分");
        System.out.println(cal.get(Calendar.SECOND) + "秒");
    }
================================================================================
===============
```

```
 java.util.GregorianCalendar[time=1646915482381,areFieldsSet=true,areAllFields
Set=true,lenient=true,zone=sun.util.calendar.ZoneInfo[id="Asia/Shanghai",offse
t=28800000,dstSavings=0,useDaylight=false,transitions=29,lastRule=null],firstD
ayOfWeek=1,minimalDaysInFirstWeek=1,ERA=1,YEAR=2022,MONTH=2,WEEK_OF_YEAR=11,WE
EK_OF_MONTH=2,DAY_OF_MONTH=10,DAY_OF_YEAR=69,DAY_OF_WEEK=5,DAY_OF_WEEK_IN_MONT
H=2,AM_PM=1,HOUR=8,HOUR_OF_DAY=20,MINUTE=31,SECOND=22,MILLISECOND=381,ZONE_OFF
SET=28800000,DST_OFFSET=0]
2022年
3月
今年第11周
本月第2周
10日
今年第69天
本周第5天
20点
31分
22秒

================================================================================
==========
    @Test
    public void test2() {
        final Calendar cal = Calendar.getInstance();//获取一个日历对象
        System.out.println(cal.get(Calendar.HOUR_OF_DAY));//获取日历对象中本月第几
天
        cal.add(Calendar.HOUR_OF_DAY, 3);//将日历对象中的小时数加三
        System.out.println(cal.get(Calendar.HOUR_OF_DAY));//
        cal.set(Calendar.HOUR_OF_DAY, 0);//将日历对象中的小时数设置为零
        System.out.println(cal.get(Calendar.HOUR_OF_DAY));
        final Date date = cal.getTime();//根据该日历对象信息获取日期对象
        System.out.println(date);
        final long timeInMillis = cal.getTimeInMillis();//根据日历信息获取时间毫秒
值
        System.out.println(timeInMillis);
    }

================================================================================
=====
20
23
0
Thu Mar 10 00:37:30 CST 2022
1646843850105
```

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

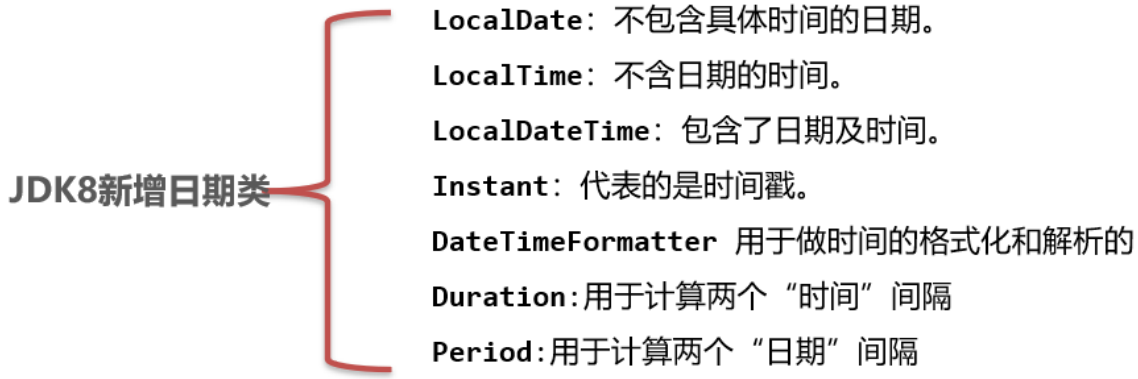
三、JDK8新增日期类

1. 从Java 8开始，java.time包提供了新的日期和时间API

1. 新增的API严格区分了时刻、本地日期、本地时间，并且，对日期和时间进行运算更加方便

1. 新API的类型几乎全部是不变类型（和String的使用类似），可以放心使用不必担心被修改。

jdk新增的七个处理日期类



LocalDate、LocalTime、LocalDateTime

他们 分别表示日期，时间，日期时间对象，他们的类的实例是不可变的对象

他们三者构建对象和API都是通用的

通用获取对象方法

构建对象的方式如下：

方法名	说明	
public static Xxxx now();	静态方法，根据当前时间创建对象	LocaDate localDate = LocalDate.now(); LocalTime llocalTime = LocalTime.*now*(); LocalDateTime localDateTime = LocalDateTime.*now*();
public static Xxxx of(...);	静态方法，指定日期/时间创建对象	LocalDate localDate1 = LocalDate.*of*(2099 , 11,11); LocalTime localTime1 = LocalTime.*of*(11, 11, 11); LocalDateTime localDateTime1 = LocalDateTime.*of*(2020, 10, 6, 13, 23, 43);

LocalDate, LocalTime, LocalDateTime获取信息的API.

LocalDate、LocalTime、LocalDateTime获取信息的API.

方法名	说明
public int getYear()	获取年
public int getMonthValue()	获取月份（1-12）
Public int getDayOfMonth()	获取月中第几天
Public int getDayOfYear()	获取年中第几天
Public DayOfWeek getDayOfWeek()	获取星期

LocalDateTime的转换API

方法名	说明
public LocalDate toLocalDate()	转换成一个LocalDate对象
public LocalTime toLocalTime()	转换成一个LocalTime对象

```
/**
     * ClassName: NewDemo
     * Description:LocalDate
     * date:2022/3/10
     *
     * @author fgcy
     * @since JDK 1.8
     */

    @Test
```

```
    public void test1() {
        final LocalDate localDate = LocalDate.now();
        System.out.println(localDate);
        System.out.println(localDate.getYear() + "年");
        System.out.println(localDate.getMonth());//MARCH
        System.out.println(localDate.getMonthValue() + "月");
        System.out.println(localDate.getDayOfMonth() + "日");
        System.out.println("今年第" + localDate.getDayOfYear() + "天");
        System.out.println(localDate.getDayOfWeek());//THURSDAY
        System.out.println("----------------");
        final LocalDate localDate1 = LocalDate.of(2023, 11, 11);
        System.out.println(localDate1);
        final LocalDate localDate2 = LocalDate.of(2023, Month.NOVEMBER, 11);//
枚举
        System.out.println(localDate2);
    }
```

```
============================================================
2022-03-10
2022年
MARCH
3月
10日
今年第69天
THURSDAY
---------------
2023-11-11
2023-11-11
```

```
    /**
     * ClassName: NewDemo
     * Description:localTime
     * date:2022/3/10
     *
     * @author fgcy
     * @since JDK 1.8
     */

    @Test
    public void test2() {
        final LocalTime localTime = LocalTime.now();
        System.out.println(localTime);
        System.out.println(localTime.getHour() + "时");
        System.out.println(localTime.getMinute() + "分");
        System.out.println(localTime.getSecond() + "秒");
        System.out.println(localTime.getNano() + "纳秒");
        System.out.println("---------------------");
        final LocalTime of = LocalTime.of(21, 21, 21, 21);
        System.out.println(of);
        System.out.println(LocalTime.of(21, 12));
```

```
        System.out.println(LocalTime.of(21, 21, 21));
    }
21:51:08.416
21时
51分
8秒
416000000纳秒
-----------------
21:21:21.000000021
21:12
21:21:21
-------------------------------------------------------------------------------
---------------------------    /**
     * ClassName: NewDemo
     * Description:LocalDateTime
     * date:2022/3/10
     *
     * @author fgcy
     * @since JDK 1.8
     */

    @Test
    public void test3() {
        final LocalDateTime localDateTime = LocalDateTime.now();
        System.out.println(localDateTime);
        System.out.println(localDateTime.getYear() + "年");
        System.out.println(localDateTime.getMonthValue() + "月");
        System.out.println(localDateTime.getDayOfMonth() + "日");
        System.out.println(localDateTime.getDayOfWeek());
        System.out.println("今年第" + localDateTime.getDayOfYear() + "天");
        System.out.println(localDateTime.getHour() + "时");
        System.out.println(localDateTime.getMinute() + "分");
        System.out.println(localDateTime.getSecond() + "秒");
        System.out.println(localDateTime.getNano() + "纳秒");
        System.out.println("--------------------------");
        final LocalDate localDate = localDateTime.toLocalDate();
        System.out.println(localDate);

        final LocalTime localTime = localDateTime.toLocalTime();
        System.out.println(localTime);

        System.out.println("------------------------");
        final LocalDateTime of = LocalDateTime.of(2021, 11, 11, 11, 11, 11);
        System.out.println(of);
    }

================================================================================
==
2022-03-10T21:54:29.027
2022年
3月
10日
THURSDAY
今年第69天
```

```
21时
54分
29秒
27000000纳秒
--------------------
2022-03-10
21:54:29.027
--------------------
2021-11-11T11:11:11
```

修改相关的API

方法名	说明
plusDays, plusWeeks, plusMonths, plusYears	向当前 LocalDate 对象添加几天、 几周、几个月、几年
minusDays, minusWeeks, minusMonths, minusYears	从当前 LocalDate 对象减去几天、 几周、几个月、几年
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 LocalDate 对象
isBefore, isAfter	比较两个 LocalDate

修改相关的API

LocalDateTime 综合了 LocalDate 和 LocalTime 里面的方法，所以只用 LocalDate 和 LocalTime 来举例。

这些方法返回的是一个新的实例引用，因为LocalDateTime 、LocalDate 、LocalTime 都是不可变的。

```java
/**
 * ClassName: NewDemo
 * 日期时间比较操作
 * Description:
 * date:2022/3/10
 *
 * @author fgcy
 * @since JDK 1.8
 */
@Test
public void test4() {
    final LocalTime localTime = LocalTime.now();
    System.out.println(localTime);
    System.out.println(localTime.minusHours(1));
    System.out.println(localTime.minusMinutes(1));
    System.out.println(localTime.minusSeconds(1));
    System.out.println(localTime.minusNanos(1));
    System.out.println("---------------------");
```

```
        System.out.println(localTime.plusHours(1));
        System.out.println(localTime.plusMinutes(1));
        System.out.println(localTime.plusSeconds(1));
        System.out.println(localTime.plusNanos(1));
        System.out.println(localTime);
        System.out.println("----------------------");
        final LocalDate of = LocalDate.of(2023, 11, 5);
        final LocalDate now = LocalDate.now();
        System.out.println("今天是2023-11-5吗？" + of.equals(now));
        System.out.println("今天在2023-11-5之前？" + now.isBefore(of));
        System.out.println("今天在2023-11-5之后？" + now.isAfter(of));
        System.out.println("----------------------");
        final LocalDate birthDate = LocalDate.of(2021, 3, 10);
        final MonthDay birthMonthDay = MonthDay.of(birthDate.getMonthValue(),
birthDate.getDayOfMonth());//通过月日的数值获取月日对象
        final MonthDay nowMonthDay = MonthDay.from(now);//通过LocalDate对象获得月
日对象
        System.out.println("今天是你的生日吗？" +
birthMonthDay.equals(nowMonthDay));
    }
=====================================================
23:02:54.866
22:02:54.866
23:01:54.866
23:02:53.866
23:02:54.865999999
----------------------
00:02:54.866
23:03:54.866
23:02:55.866
23:02:54.866000001
23:02:54.866
----------------------
今天是2023-11-5吗？false
今天在2023-11-5之前？true
今天在2023-11-5之后？false
----------------------
今天是你的生日吗？true
```

Instant时间戳

JDK8获取时间戳特别简单，且功能更丰富。Instant类由一个静态的工厂方法now()可以返回当前时间戳时间戳是包含日期和时间的，与java.util.Date很类似，事实上Instant就是类似JDK8 以前的Date。Instant和Date这两个类可以进行转换;Date.from(instant); date.toInstant();

```
    @Test
    public void test1() {
        final Instant instant = Instant.now();
        System.out.println(instant);//重写了toString方法，输出默认世界时间
```

```
        System.out.println(instant.atZone(ZoneId.systemDefault()));//改为系统默认时区，输出上海时间(返回值类型：ZonedDateTime)

 System.out.println(Instant.ofEpochMilli(System.currentTimeMillis()));//根据时间毫秒值获取Instant对象

        final Date date = Date.from(instant);//将Instant对象转为Date对象
        System.out.println(date);
        final Instant toInstant = date.toInstant();//将Date对象转为Instant对象
        System.out.println(toInstant);
    }
================================================================================
==================
2022-03-10T15:22:37.686Z
2022-03-10T23:22:37.686+08:00[Asia/Shanghai]
2022-03-10T15:22:37.728Z
Thu Mar 10 23:22:37 CST 2022
2022-03-10T15:22:37.686Z
```

DateTimeFormatter

```
    @Test
    public void test2() {
        final DateTimeFormatter ofPattern = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss EEE a");
        final LocalDateTime localDateTime = LocalDateTime.now();
        final String format = ofPattern.format(localDateTime);//正向
        System.out.println(format);
        System.out.println(localDateTime.format(ofPattern));//反向

        final LocalDateTime localDateTime1 = LocalDateTime.parse("2022-03-10 23:35:12 星期四 下午", ofPattern);
        System.out.println("今天是今年的第" + localDateTime1.getDayOfYear() + "天");
    }

================================================================================
=============
2022-03-10 23:36:31 星期四 下午
2022-03-10 23:36:31 星期四 下午
今天是今年的第69天
```

Period

1. 在Java8中，我们可以使用以下类来计算日期间隔差异：java.time.Period
2. 主要是 Period 类方法 getYears()，getMonths() 和 getDays() 来计算,只能精确到年月日。
3. 用于 LocalDate 之间的比较

```
    @Test
    public void test3() {
        final LocalDate now = LocalDate.now();
        final LocalDate birthday = LocalDate.of(2000, 3, 2);
        final Period period = Period.between(birthday, now);//前小后大
        //模板字符串 类C printf()
        System.out.printf("到目前为止，你已经活了%d年零%d个月零%d天",
period.getYears(), period.getMonths(), period.getDays());
    }
```

```
==========
    到目前为止，你已经活了22年零0个月零8天
```

Duration

1. 在Java8中，我们可以使用以下类来计算时间间隔差异：java.time.Duration
2. 提供了使用基于时间的值测量时间量的方法。
3. 用于 LocalDateTime 之间的比较。也可用于 Instant 之间的比较

```
    @Test
    public void test4() {
        final LocalDateTime localDateTime = LocalDateTime.now();
        final LocalDateTime birthDateTime = LocalDateTime.of(2000, 3, 2, 14,
0, 0);
        final Duration duration = Duration.between(birthDateTime,
localDateTime);
        System.out.printf("你到目前为止已经活了%d天即%d分钟即%d毫秒",
duration.toDays(), duration.toMinutes(), duration.toMillis());
    }
```

```
==========
    你到目前为止已经活了8043天即11582522分钟即694951364440毫秒
```

1、 Duration: 用于计算两个“时间”间隔。
2、 Period: 用于计算两个“日期”间隔

ChronoUnit

ChronoUnit类可用于在单个时间单位内测量一段时间，这个工具类是最全的了，可以用于比较所有的时间单位
java.time.temporal.ChronoUnit

```
  @Test
    public void test5() {
```

```
        final LocalDateTime today = LocalDateTime.now();
        final LocalDateTime birthDate = LocalDateTime.of(2000, 3, 2, 14, 0,
0);
        System.out.println("相差的年数：" + ChronoUnit.YEARS.between(birthDate,
today));
        System.out.println("相差的月数：" + ChronoUnit.MONTHS.between(birthDate,
today));
        System.out.println("相差的周数：" + ChronoUnit.WEEKS.between(birthDate,
today));
        System.out.println("相差的天数：" + ChronoUnit.DAYS.between(birthDate,
today));
        System.out.println("相差的时数：" + ChronoUnit.HOURS.between(birthDate,
today));
        System.out.println("相差的分数：" + ChronoUnit.MINUTES.between(birthDate,
today));
        System.out.println("相差的秒数：" + ChronoUnit.SECONDS.between(birthDate,
today));
        System.out.println("相差的毫秒数：" +
ChronoUnit.MILLIS.between(birthDate, today));
        System.out.println("相差的微秒数：" +
ChronoUnit.MICROS.between(birthDate, today));
        System.out.println("相差的纳秒数：" + ChronoUnit.NANOS.between(birthDate,
today));
        System.out.println("相差的半天数：" +
ChronoUnit.HALF_DAYS.between(birthDate, today));
        System.out.println("相差的十年数：" +
ChronoUnit.DECADES.between(birthDate, today));
        System.out.println("相差的世纪（百年）数：" +
ChronoUnit.CENTURIES.between(birthDate, today));
        System.out.println("相差的千年数：" +
ChronoUnit.MILLENNIA.between(birthDate, today));
        System.out.println("相差的纪元数：" + ChronoUnit.ERAS.between(birthDate,
today));
    }
```

四、包装类

概念

就是8种基本数据类型对应的引用类型。

基本数据类型	引用数据类型
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean

特点

1.Java**为了实现一切皆对象**，为8种基本类型提供了对应的引用类型。
2.后面的**集合和泛型其实也只能支持包装类型**，不支持基本数据类型.
3.**自动拆箱**：包装类型的变量可以直接赋值给基本数据类型的变量
4.**自动装箱**：基本类型的数据和变量可以直接赋值给包装类型的变量

独有功能

1.包装类的变量的**默认值可以是null**，**容错率更高**

2.可以把基本类型的数据转换成字符串类型(用处不大)
调用toString()方法得到字符串结果。
调用Integer.toString(基本类型的数据)

3.可以**把字符串类型的数值转换成真实的数据类型**（真的很有用）
Integer.parseInt(“字符串类型的整数”)
Double.parseDouble(“字符串类型的小数”)
Integer.valueOf("字符串类型的整数")
Double.valueOf("字符串类型的小数")

```
/**
 * ClassName: WrapDemo
 * Description:z自动装箱，拆箱
 * date:2022/3/11
 *
 * @author fgcy
```

```
     * @since JDK 1.8
     */

    @Test
    public void test1() {
        Integer a = 1;
        int b = a;//自动拆箱
        System.out.println(b);
        System.out.println("----------------");
        int c = 2;
        Integer d = c;//自动装箱
        System.out.println(d);
    }
================================================================================
============
1
----------------
2

--------------------------------------------------------------------------------
----------------------
        @Test
    public void test2() {
        //int a=null;基本数据类型不能为null
        Integer b = null;//包装类时对象可以为null
        Integer integer = 12;
        System.out.println(integer.toString() + 1);//数字转字符串1
        System.out.println(Integer.toString(integer) + 1);//数字转字符串2
        System.out.println(integer + "" + 1);//数字转字符串3
        System.out.println("------------------------");
        System.out.println(Integer.parseInt("123") + 1);//字符串转数字1
        System.out.println(Integer.valueOf("123") + 1);//字符串转数字2
        System.out.println(Double.valueOf("3.14") + 0.01);//字符串转小数1
        System.out.println(Double.parseDouble("3.14") + 0.01);//字符串转小数2
    }
================================================================================
==========
121
121
121
------------------------
124
124
3.15
3.15
```

总结

1、包装类是什么，作用是什么?
基本数据类型对应的引用类型，实现了一切皆对象。
后期集合和泛型不支持基本类型，只能使用包装类。

2、包装类有哪些特殊功能?
可以把基本类型的数据转换成字符串类型(用处不大)
可以把字符串类型的数值转换成真实的数据类型（真的很有用）

注意：(String) String.valueOf() toString区别

五、正则表达式

概念

正则表达式可以用一些规定的字符来制定规则，并用来校验数据格式的合法性

匹配字符的规则

使用详解

字符串对象提供了匹配正则表达式的方法

```
public boolean matches(String regex)
```

判断是否匹配正则表达式，匹配返回true，不匹配返回false

字符类(默认匹配一个字符)

[abc]	只能是a, b, 或c
[^abc]	除了a, b, c之外的任何字符
[a-zA-Z]	a到z A到Z，包括（范围）
[a-d[m-p]]	a到d，或m通过p：（[a-dm-p]联合）
[a-z&&[def]]	d, e, 或f(交集)
[a-z&&[^bc]]	a到z，除了b和c：（[ad-z]减法）
[a-z&&[^m-p]]	a到z，除了m到p：（[a-lq-z]减法）

预定义的字符类(默认匹配一个字符)

.	任何字符
\d	一个数字：[0-9]
\D	非数字：[^0-9]
\s	一个空白字符：[\t\n\x0B\f\r]
\S	非空白字符：[^\s]
\w	[a-zA-Z_0-9] 英文、数字、下划线
\W	[^\w] 一个非单词字符

贪婪的量词（配合匹配多个字符）

X?	X，一次或根本不
X*	X，零次或多次
X+	X，一次或多次
X {n}	X，正好n次
X {n, }	X，至少n次
X {n,m}	X，至少n但不超过m次

案例

```
private static void checkMail() {
    final Scanner sc = new Scanner(System.in);
    while (true) {
        System.out.print("请输入邮箱号:");
        final String mail = sc.nextLine();
        //zwj1061499050@126.com
```

```
            if (mail.matches("\\w{1,20}@[a-zA-Z0-9]{2,4}(\\.[a-z]{2,3})
{1,2}")) {
                System.out.println("邮箱验证成功！");
                break;
            } else {
                System.out.println("邮箱格式有误");
            }
        }
    }
```

```
==========================================================
请输入邮箱号:@qq.com
邮箱格式有误
请输入邮箱号:132.fffd@fff.co
邮箱格式有误
请输入邮箱号:zwj1061499050@126.com.cn
邮箱验证成功!
```

```
-------------------------------------------------------------------------------
------------------------
        private static void checkPhone() {
        final Scanner sc = new Scanner(System.in);
        while (true) {
            System.out.print("请输入手机号:");
            final String phone = sc.nextLine();
            if (phone.matches("1[3-9]\\d{9}")) {
                System.out.println("手机验证成功！");
                break;
            } else {
                System.out.println("手机格式有误");
            }
        }
    }
```

```
=============================================================================
====
请输入手机号:123456789111
手机格式有误
请输入手机号:12345678911
手机格式有误
请输入手机号:aaaaaaaaaaa
手机格式有误
请输入手机号:17665661294
手机验证成功!
```

正则表达式在字符串中的使用

方法名	说明
public String replaceAll(String regex,String newStr)	按照正则表达式匹配的内容进行替换
public String[] split(String regex):	按照正则表达式匹配的内容进行分割字符串，反回一个字符串数组。

```
    @Test
    public void test1() {
        final String data = "闪避908_呼叫544ffff你爸妈14jkl吧那";
        final String[] strings = data.split("\\w+");
        for (String string : strings) {
            System.out.println(string);
        }
        final String s = data.replaceAll("\\w+", ",");
        System.out.println(s);
    }
```

```
==========================================================================
闪避
呼叫
你爸妈
吧那
闪避,呼叫,你爸妈,吧那
```

正则表达式爬取信息

```
    @Test
    public void test2() {
        final String data = "来黑马程序学习Java,电话020-43422424，或者联系邮箱" +
                "itcast@itcast.cn,电话18762832633，0203232323" +
                "邮箱bozai@itcast.cn，400-100-3233 ， 4001003232";

        final Pattern pattern = Pattern.compile("(\\w{1,20}@[a-zA-Z0-9]{2,10}
(\\.[a-z]{2,3}){1,2})|" +
                "(1[3-9]\\d{9})|" +
                "(0\\d{2,5}-?\\d{5,15})|(400-?\\d{3,8}-?\\d{3,8})");

        final Matcher matcher = pattern.matcher(data);
        while (matcher.find()){
            System.out.println(matcher.group());
        }
    }
```

```
==========================================================
020-43422424
itcast@itcast.cn
18762832633
0203232323
bozai@itcast.cn
400-100-3233
4001003232
```

操作数组元素

Arrays概念

数组操作工具类，专门用于操作数组元素的

Arrays类的常用api

Arrays类的常用API

方法名	说明
public static String toString(类型[] a)	返回数组的内容（字符串形式）
public static void sort(类型[] a)	对数组进行默认升序排序
public static <T> void sort(类型[] a, Comparator<? super T> c)	使用比较器对象自定义排序
public static int binarySearch(int[] a, int key)	二分搜索数组中的数据，存在返回索引，不存在返回-1

Arrays类常用功能演示

```
    @Test
    public void test1() {
        final double[] doubles = {1.0, 42.5, 5, 4.21, 55, 17.5, 11.33};
        System.out.println(Arrays.toString(doubles));//将数组转为字符串
        Arrays.sort(doubles);//默认升序排序
        System.out.println(doubles);
        System.out.println(Arrays.toString(doubles));
        //二分排序，前提有序，找到返回索引，找不到返回它应该插入的位置+1，然后总体取负数
        System.out.println(Arrays.binarySearch(doubles, 12));
        final int[] ints = {12, 44, 5, 32, 84, -4, 122};
        //如果二分查找前，数组并没有排好序，也不会报错，就是结果不准确
        System.out.println(Arrays.binarySearch(ints, 44));//应该时1，结果是-7
    }
==================================================
[1.0, 42.5, 5.0, 4.21, 55.0, 17.5, 11.33]
[D@3d82c5f3
[1.0, 4.21, 5.0, 11.33, 17.5, 42.5, 55.0]
-5
-7
```

Arrays类的排序

方法名	说明
public static void sort(类型[] a)	对数组进行默认升序排序
public static <T> void sort(类型[] a, Comparator<? super T> c)	使用比较器对象自定义排序

自定义排序规则

设置Comparator接口对应的比较器对象，来定制比较规则

如果认为左边数据 大于 右边数据 返回正整数

如果认为左边数据 小于 右边数据 返回负整数

如果认为左边数据 等于 右边数据 返回0

(o1-o2)升序排序，(o2-o1)降序排序

```
--------------------------------------------------------------------------------
-------------------------
     @Test
    public void test2() {
        final Integer[] ints = {12, 55, 3, -7, 54, 34, 91, 4};
/*        Arrays.sort(ints, new Comparator<Integer>() {
            @Override
            public int compare(Integer o1, Integer o2) {
                return o2 - o1;
            }
        });*/
        Arrays.sort(ints, (o1, o2) -> o2 - o1);
        System.out.println(Arrays.toString(ints));
================================================================
[91, 55, 54, 34, 12, 4, 3, -7]

--------------------------------------------------------------------------------
-------------------------
        class Student {
        private String name;
        private double height;
        private int age;


        public Student() {
        }

        public Student(String name, double height, int age) {
            this.name = name;
            this.height = height;
            this.age = age;
        }

    }

    @Test
```

```
    public void test4() {
        final Student[] students = new Student[3];
        students[0] = new Student("张三", 177, 15);
        students[1] = new Student("未晓", 175, 18);
        students[2] = new Student("初沐", 165, 20);
        Arrays.sort(students, (o1, o2) -> o2.getAge() - o1.getAge());
        System.out.println(Arrays.toString(students));
    }
```

```
================================================================================
[Student{name='初沐', height=165.0, age=20},
 Student{name='未晓', height=175.0, age=18},
 Student{name='张三', height=177.0, age=15}]
```

常见算法

冒泡排序

冒泡排序的思想

每次从数组中找出最大值放在数组的后面去

实现步骤：

确定总共需要做几轮： 数组的长度-1.

每轮比较几次

当前位置大于后一个位置则交换数据

```
    /*
     * 0 1 2 3
     * 0   012
     * 1   01
     * 2   0
     *
     * 一共有四个数，经过每轮比较得到一个最大或最小的数；一共需要比较4-1次
     * 每次比较都要确保除了已经比较完的最值外，其他所有数都要比较一次
     * */
    public void bubbleSorted(int[] arr) {
        if (arr == null) {
            System.out.println("null");
            return;
        }
        if (arr.length <= 0) {
            System.out.println("空数组");
            return;
        }
        for (int i = 0; i < arr.length - 1; i++) {
            for (int j = 0; j < arr.length - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
        System.out.println(Arrays.toString(arr));
```

```
    }


    @Test
    public void test1() {
        final int[] ints = {12, 21, 43, 6, 94, -4, 0, 33};
        bubbleSorted(ints);
        bubbleSorted(null);
        bubbleSorted(new int[0]);
    }
================================================================================
=======
[-4, 0, 6, 12, 21, 33, 43, 94]
null
空数组
```

选择排序

选择排序的思想

每轮选择当前位置，开始找出后面的较小值与该位置交换

实现步骤：

确定总共需要选择几轮： 数组的长度-1.

控制每轮从以前位置为基准，与后面元素选择几次

```
    /*
     *
     * 0 1 2 3 4
     *
     *
     * 0 1234
     * 1 234
     * 2 34
     * 3 4
     * */
    public void selectSorted(int[] arr) {
        if (arr == null) {
            System.out.println("null");
            return;
        }
        if (arr.length <= 0) {
            System.out.println("空数组");
            return;
        }
        for (int i = 0; i < arr.length - 1; i++) {
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[i] > arr[j]) {
                    int temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                }
```

```
            }
        }
        System.out.println(Arrays.toString(arr));
    }


    @Test
    public void test1() {
        final int[] ints = {12, 21, 43, 6, 94, -4, 0, 33};
        selectSorted(ints);
        selectSorted(null);
        selectSorted(new int[0]);
    }
================================================================================
[-4, 0, 6, 12, 21, 33, 43, 94]
null
空数组
```

二分查找

1.在数据量特别大的时候，基本查找从前往后寻找的性能是很差的
2.二分查询性能好，二分查找的前提是必须是排好序的数据
3.二分查找相当于每次去掉一半的查找范围

二分查找实现步骤

定义变量记录左边和右边位置。
使用while循环控制查询（条件是左边位置<=右边位置）
循环内部获取中间元素索引
判断当前要找的元素如果大于中间元素，左边位置=中间索引+1
判断当前要找的元素如果小于中间元素，右边位置=中间索引-1
判断当前要找的元素如果等于中间元素，返回当前中间元素索引。

```
    @Test
    public void test5() {
        final int[] ints = {11, 13, 23, 27, 34, 65, 88};
        System.out.println(binarySearch(ints, 27));
    }

    public int binarySearch(int[] arr, int target) {
        if (arr == null || arr.length <= 0) {
            return -1;
        }
        int left = 0;
        int right = arr.length - 1;
```

```
        while (left <= right) {
            int mid = (left + right) / 2;
            if (target > arr[mid]) {
                left = mid + 1;
            } else if (target < arr[mid]) {
                right = mid - 1;
            } else {
                return mid;
            }
        }
        return -1;
    }
==============================================
3
```

Lambda表达式

Lambda表达式的概述

Lambda表达式是JDK 8开始后的一种新语法形式

作用：**简化匿名内部类的代码写法**

Lambda表达式的简化格式

```
(匿名内部类被重写方法的形参列表) -> {
    被重写方法的方法体代码。
}
```

注：-> 是语法形式，无实际含义

注意：Lambda表达式**只能简化函数式接口的匿名内部类**的写法形式

什么是函数式接口
首先必须是接口、其次**接口中有且仅有一个抽象方法**的形式

代码更少，关注点更加明确了

注意：通常我们见到的函数式接口上都有一个**@FunctionalInterface注解**，标记该接口必须是满足函数式接口

```
package lambda_demo;


public class LambdaDemo {
    public static void main(String[] args) {
        Swimming s1 = new Swimming() {
            @Override
            public void swim() {
                System.out.println("老师游泳");
            }
        };
        Swimming s2 = () -> System.out.println("学生游泳");
        go(s1);
        go(s2);
    }

    public static void go(Swimming swimming) {
        System.out.println("开始");
        swimming.swim();
        System.out.println("结束");
    }
}

//该注解代表该接口时函数式接口即只有一个抽象方法，多写一个方法就会报错；Lambda表达式可以简化函数式接口；
//不加注解也可以简化，前提是，该接口只有一个抽象方法
@FunctionalInterface
interface Swimming {
    void swim();
}
========================================================================
开始
老师游泳
结束
开始
学生游泳
结束
```

总结

Lambda表达式的基本作用：
简化匿名内部类的写法

Lambda表达式的使用前提：
1.函数式接口，即是一个对象且对象中只有一个抽象方法
2.通过匿名内部类的方式创建对象

Lambda表达式的好处：

简化代码，使得java语言的表达能力得到提高

- Lambda表达式的省略规则

  参数类型可以省略不写。
  如果只有一个参数，参数类型可以省略，同时()也可以省略。
  如果Lambda表达式的方法体代码只有一行代码。可以省略大括号不写,同时要省略分号！
  如果Lambda表达式的方法体代码只有一行代码。可以省略大括号不写。

  此时，如果这行代码是return语句，【必须】省略return不写，同时也必须省略 ";"