16.常用工具类介绍

【本节目标】

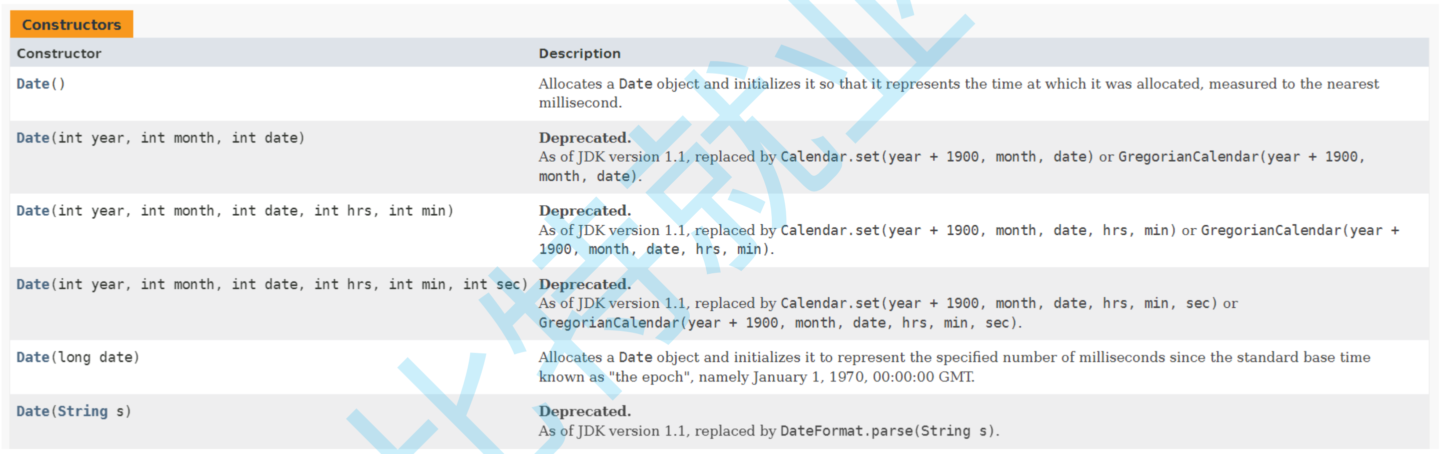
- 熟悉日期相关类
- 熟悉BigDecimal类

1. 日期相关类

1.1 Date类

Date类是Java早期版本中用于表示日期和时间的类，位于 `java.util` 包中。它表示一个特定的日期和时间，精确到毫秒级别。官方提供的帮助文档手册：Class Date

常见的构造方法如下：



Constructors

Constructor	Description
`Date()`	Allocates a `Date` object and initializes it so that it represents the time at which it was allocated, measured to the nearest millisecond.
`Date(int year, int month, int date)`	**Deprecated.** As of JDK version 1.1, replaced by `Calendar.set(year + 1900, month, date)` or `GregorianCalendar(year + 1900, month, date)`.
`Date(int year, int month, int date, int hrs, int min)`	**Deprecated.** As of JDK version 1.1, replaced by `Calendar.set(year + 1900, month, date, hrs, min)` or `GregorianCalendar(year + 1900, month, date, hrs, min)`.
`Date(int year, int month, int date, int hrs, int min, int sec)`	**Deprecated.** As of JDK version 1.1, replaced by `Calendar.set(year + 1900, month, date, hrs, min, sec)` or `GregorianCalendar(year + 1900, month, date, hrs, min, sec)`.
`Date(long date)`	Allocates a `Date` object and initializes it to represent the specified number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT.
`Date(String s)`	**Deprecated.** As of JDK version 1.1, replaced by `DateFormat.parse(String s)`.

我们可以看到部分方法已经过时。

```java
public static void main(String[] args) {
    Date date1 = new Date();
    System.out.println(date1);
    //已过时
    Date date2 = new Date(123,2,5);
    System.out.println(date2);
}
```

`Date date2 = new Date(123,2,5);` 该方法中，需要注意参数的意义：

1. 参数一：Date默认的时间是从1900年开始计算的，这里的123会和1900相加得到2023，用来确定年份

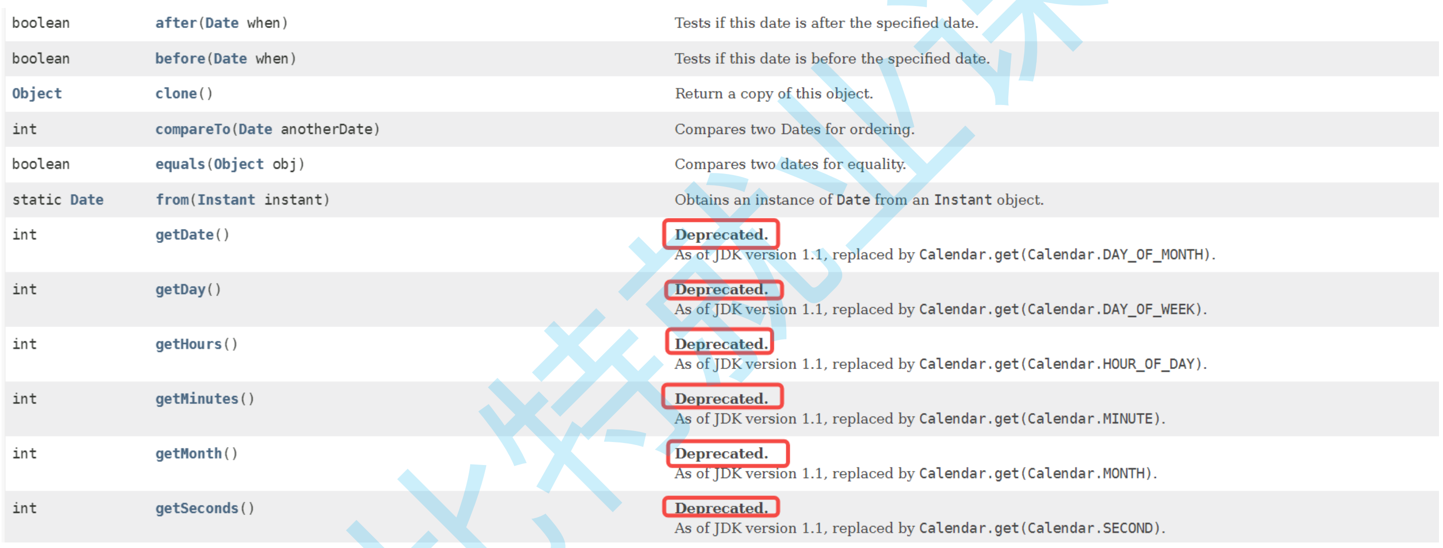
2. 参数二：2代表3月，也就是说0代表1月，1代表2月，以此类推
3. 参数三：代表实际的日期

输出结果：

```
Wed Oct 09 15:30:46 CST 2024
Sun Mar 05 00:00:00 CST 2023
```

- CST：代表时区
- 不带参数的构造方法表示获取的是当前的时间如：15:30:46
- 给定参数后，没有指定时间，默认是00:00:00

当我们查看源码后发现，很多方法都已经过时了。目前我们使用更多的是LocalDateTime类。所以，我们重点要看看这个类！！



boolean	after(Date when)	Tests if this date is after the specified date.
boolean	before(Date when)	Tests if this date is before the specified date.
Object	clone()	Return a copy of this object.
int	compareTo(Date anotherDate)	Compares two Dates for ordering.
boolean	equals(Object obj)	Compares two dates for equality.
static Date	from(Instant instant)	Obtains an instance of Date from an Instant object.
int	getDate()	Deprecated. As of JDK version 1.1, replaced by Calendar.get(Calendar.DAY_OF_MONTH).
int	getDay()	Deprecated. As of JDK version 1.1, replaced by Calendar.get(Calendar.DAY_OF_WEEK).
int	getHours()	Deprecated. As of JDK version 1.1, replaced by Calendar.get(Calendar.HOUR_OF_DAY).
int	getMinutes()	Deprecated. As of JDK version 1.1, replaced by Calendar.get(Calendar.MINUTE).
int	getMonth()	Deprecated. As of JDK version 1.1, replaced by Calendar.get(Calendar.MONTH).
int	getSeconds()	Deprecated. As of JDK version 1.1, replaced by Calendar.get(Calendar.SECOND).

1.2 LocalDateTime类

`LocalDateTime` 是**Java 8**引入的日期类。官方手册：Class LocalDateTime

`LocalDateTime` 类只有一个私有的构造方法，共其内部进行调用。

```
private LocalDateTime(LocalDate date, LocalTime time) {
    this.date = date;
    this.time = time;
}
```

1.2.1 创建LocalDateTime 对象

```java
// 当前日期和时间
 LocalDateTime now = LocalDateTime.now();
// 指定日期和时间
LocalDateTime dateTime = LocalDateTime.of(2023, 5, 15, 10, 30);
// 从字符串解析
LocalDateTime parsed = LocalDateTime.parse("2023-05-15T10:30:00");
```

1.2.2 常用方法-获取当前日期

```java
public static void main(String[] args) {
    LocalDateTime dateTime  = LocalDateTime.now();
    System.out.println("当前时间为: "+dateTime );
}
//输出结果:
当前日期为: 2024-10-15T14:17:06.747600200
```

1.2.3 常用方法-获取当前年月日

```java
public static void main(String[] args) {
    LocalDateTime dateTime = LocalDateTime.now();

    int year = dateTime.getYear();
    int month = dateTime.getMonthValue();
    int day = dateTime.getDayOfMonth();
    int hour = dateTime.getHour();
    int minute = dateTime.getMinute();
    int second = dateTime.getSecond();

    System.out.println("年: "+year);
    System.out.println("月: "+month);
    System.out.println("日: "+day);
    System.out.println("时: "+hour);
    System.out.println("分: "+minute);
    System.out.println("秒: "+second);

}
//输出结果:
年: 2024
月: 10
日: 15
时: 14
分: 19
秒: 13
```

1.2.4 常用方法-创建指定的日期

```java
public static void main(String[] args) {
    LocalDateTime dateTime = LocalDateTime.of(1999, 5, 15, 10, 30);
    System.out.println("当前日期为: "+dateTime);
}
//输出结果:
当前日期为: 1999-05-15T10:30
```

1.2.5 常用方法-根据字符串创建日期

```java
public static void main(String[] args) {
    String stringDate = "2026-10-01 10:30:21";
    //创建一个自定义的日期时间格式化器
    DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
    LocalDateTime  date = LocalDateTime .parse(stringDate,dateTimeFormatter);
    System.out.println("当前日期为: "+date);
}
//当前日期为: 2026-10-01T10:30:21
```

需要注意的是这里一定是: `yyyy-MM-dd HH:mm:ss` 大小写需要注意。

1.2.6 常用方法-获取具体详细信息

- 获取本周周几、本月第几天，当年第几天

```java
public static void main(String[] args) {
    LocalDateTime dateTime = LocalDateTime.now();
    System.out.println("当前日期为: "+dateTime);
    System.out.println("本周周几: "+dateTime.getDayOfWeek().getValue());
    System.out.println("本月第几天: "+dateTime.getDayOfMonth());
    System.out.println("当年第几天: "+dateTime.getDayOfYear());
}
```

1.2.7 常用方法-日期运算

- 增加、减少天数

```java
public static void main(String[] args) {
    LocalDateTime dateTime = LocalDateTime.now();
    LocalDateTime newDatePlus = dateTime.plusDays(1);
    System.out.println("增加1天后的日期: "+newDatePlus);

    LocalDateTime newDateMinus = dateTime.minusDays(1);
    System.out.println("减少1天后的日期: "+newDateMinus);

}
```

- 增加、减少周数

```
public static void main(String[] args) {
    LocalDateTime dateTime = LocalDateTime.now();
    LocalDateTime newDatePlus = dateTime.plusWeeks(1);
    System.out.println("增加1周后的日期: "+newDatePlus);

    LocalDateTime newDateMinus = dateTime.minusWeeks(1);
    System.out.println("减少1周后的日期: "+newDateMinus);

}
```

- 增加、减少月数

```
public static void main(String[] args) {
    LocalDateTime dateTime = LocalDateTime.now();
    LocalDateTime newDatePlus = dateTime.plusMonths(1);
    System.out.println("增加1月后的日期: "+newDatePlus);

    LocalDateTime newDateMinus = dateTime.minusMonths(1);
    System.out.println("减少1月后的日期: "+newDateMinus);

}
```

- 增加、减少年数

```
public static void main(String[] args) {
    LocalDateTime dateTime = LocalDateTime.now();
    LocalDateTime newDatePlus = dateTime.plusYears(1);
    System.out.println("增加1年后的日期: "+newDatePlus);

    LocalDateTime newDateMinus = dateTime.minusYears(1);
```

```java
    System.out.println("减少1年后的日期: "+newDateMinus);

}
```

1.2.8 常用方法-根据当前时间获取指定时间

- 获取当前日期所在周的周日和周一

```java
public static void main(String[] args) {
    LocalDateTime currentDate = LocalDateTime.now();

    //获取当前日期所在的当周周一
    LocalDateTime firstDayOfWeek = 
currentDate.with(TemporalAdjusters.previousOrSame(DayOfWeek.MONDAY));
    //获取当前日期所在的当周周日
    LocalDateTime lastDayOfWeek = 
currentDate.with(TemporalAdjusters.nextOrSame(DayOfWeek.SUNDAY));

    System.out.println(firstDayOfWeek);
    System.out.println(lastDayOfWeek);
}
```

> - previousOrSame：寻找当前日期或之前最近的指定星期几
> - nextOrSame：寻找当前日期或之后最近的指定星期几

- 获取当前日期所在月的第一天和最后一天的日期

```java
public static void main(String[] args) {
    LocalDateTime currentDate = LocalDateTime.now();

    LocalDateTime firstDayOfMonth = 
currentDate.with(TemporalAdjusters.firstDayOfMonth());

    LocalDateTime lastDayOfMonth = 
currentDate.with(TemporalAdjusters.lastDayOfMonth());

    System.out.println(firstDayOfMonth);
    System.out.println(lastDayOfMonth);
}
```

2. BigDecimal类

BigDecimal是Java在java.math包中提供的 `线程安全` 的API类，用来对超过16位有效位的数进行精确的运算。双精度浮点型变量double可以处理16位有效数，但在实际应用中，可能需要对更大或者更小的数进行运算和处理。官方文档手册：Class BigDecimal

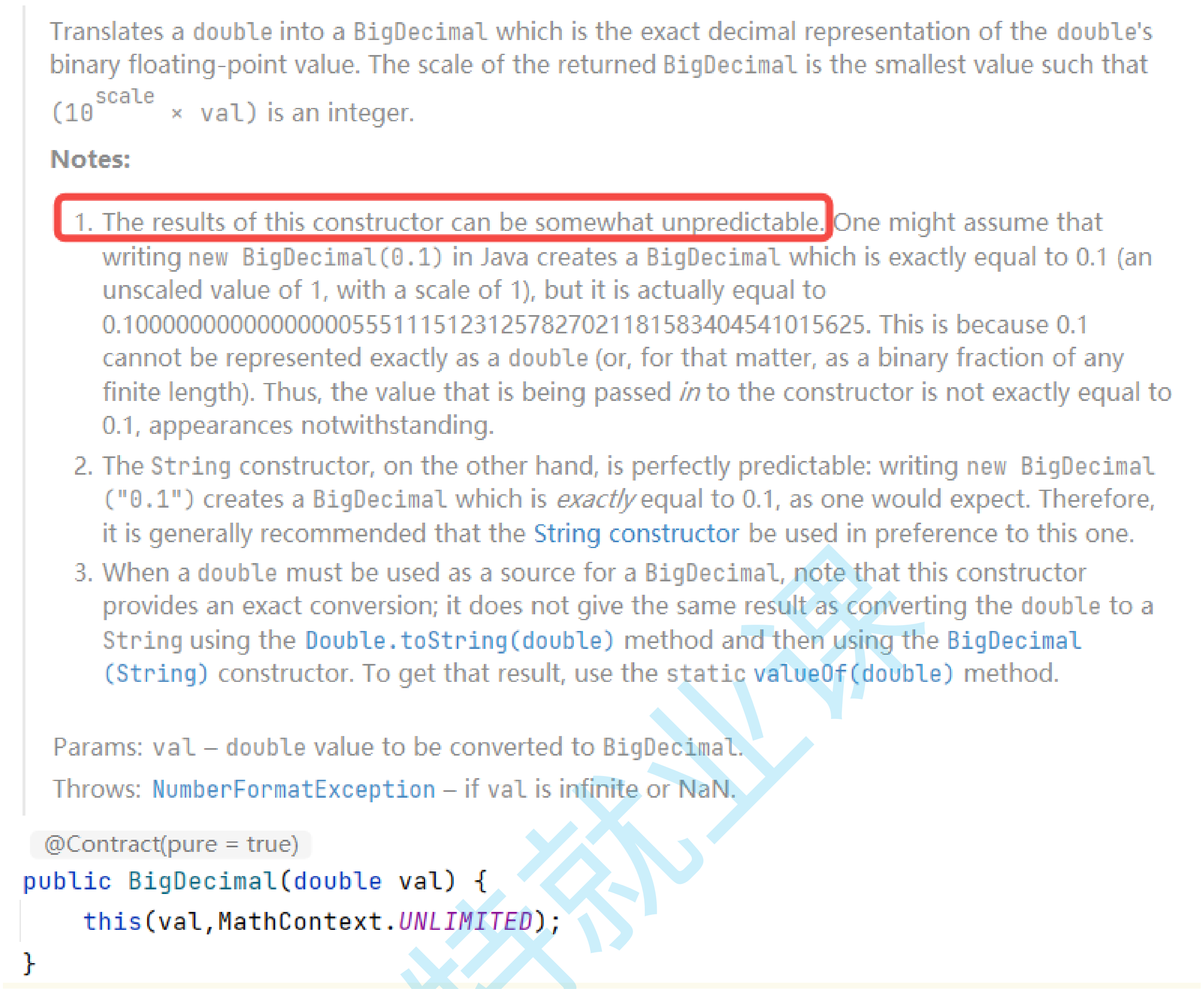
2.1 常用的构造方法

构造器	描述
BigDecimal(int)	创建一个具有参数所指定整数
BigDecimal(double)	创建一个具有参数所指定双精度值的 可能会丢失精度】
BigDecimal(long)	创建一个具有参数所指定长整
BigDecimal(String)	创建一个具有参数所指定以字符串表

```java
public static void main(String[] args) {
    BigDecimal doubleNum =new BigDecimal(1.99);
    System.out.println( doubleNum);
    BigDecimal stringNum = new BigDecimal("2.99");
    System.out.println( stringNum);
}
//输出结果:
1.9899999999999999911182158029987476766109466552734375
2.99
```

比较有意思的是，这里第一个输出的结果非常长，主要原因是使用 `new BigDecimal(1.99)` 时，实际上是将一个已经被舍入的 double 值传递给了 BigDecimal。BigDecimal 然后精确地表示了这个已经不精确的 double 值。

官方说明如下：



Translates a double into a BigDecimal which is the exact decimal representation of the double's binary floating-point value. The scale of the returned BigDecimal is the smallest value such that $(10^{scale} \times val)$ is an integer.

Notes:

1. The results of this constructor can be somewhat unpredictable. One might assume that writing new BigDecimal(0.1) in Java creates a BigDecimal which is exactly equal to 0.1 (an unscaled value of 1, with a scale of 1), but it is actually equal to 0.1000000000000000055511151231257827021181583404541015625. This is because 0.1 cannot be represented exactly as a double (or, for that matter, as a binary fraction of any finite length). Thus, the value that is being passed *in* to the constructor is not exactly equal to 0.1, appearances notwithstanding.
2. The String constructor, on the other hand, is perfectly predictable: writing new BigDecimal("0.1") creates a BigDecimal which is *exactly* equal to 0.1, as one would expect. Therefore, it is generally recommended that the String constructor be used in preference to this one.
3. When a double must be used as a source for a BigDecimal, note that this constructor provides an exact conversion; it does not give the same result as converting the double to a String using the Double.toString(double) method and then using the BigDecimal(String) constructor. To get that result, use the static valueOf(double) method.

Params: val – double value to be converted to BigDecimal.
Throws: NumberFormatException – if val is infinite or NaN.

```
@Contract(pure = true)
public BigDecimal(double val) {
    this(val,MathContext.UNLIMITED);
}
```

所以我们更推荐使用字符串的形式，创建对象。

2.2 常用方法-加减乘

```java
public static void main(String[] args) {
    BigDecimal a =new BigDecimal("1.35");
    BigDecimal b = new BigDecimal("3.22");

    BigDecimal addRet = a.add(b);
    System.out.println(addRet);

    BigDecimal subRet = a.subtract(b);
    System.out.println(subRet);

    BigDecimal mulRet = a.multiply(b);
    System.out.println(mulRet);
}
```

注意：

- 参与运算后会生成新的BigDecimal 对象

2.3 常用方法-除

```
public static void main(String[] args) {
    BigDecimal a =new BigDecimal("1.35");
    BigDecimal b = new BigDecimal("3.22");

    BigDecimal divRet = a.divide(b);
    System.out.println(divRet);

}
```

报错信息如下：

```
Exception in thread "main" java.lang.ArithmeticException: Non-terminating
decimal expansion; no exact representable decimal result.
        at java.base/java.math.BigDecimal.divide(BigDecimal.java:1780)
        at Person.main(LibrarySystem.java:122)

```

原因：BigDecimal 的 divide 方法在进行除法运算时，如果结果是一个无限循环小数，就会抛出 ArithmeticException 异常。在你的例子中，1.35 除以 3.22 的结果正是一个无限循环小数。

解决方案一：

指定精度和舍入模式

```
public static void main(String[] args) {
    BigDecimal a =new BigDecimal("1.35");
    BigDecimal b = new BigDecimal("3.22");

    BigDecimal divRet = a.divide(b, 4, RoundingMode.HALF_UP);
    System.out.println(divRet);

}
//输出结果
0.4193
```

- 4 表示除法运算结果的小数部分**保留 4 位**

- RoundingMode.HALF_UP 代表四舍五入（即 `0.5` 向上舍入）

关于第3个参数属于运算的模式。常见的有：

模式	说明
RoundingMode.UP	始终向远离零的方向舍入。 例如：1.5 → 2，-1.5 → -2
RoundingMode.DOWN	始终向接近零的方向舍入。 例如：1.5 → 1，-1.5 → -1
RoundingMode.CEILING	向正无穷大方向舍入。 例如：1.5 → 2，-1.5 → -1
RoundingMode.FLOOR	向负无穷大方向舍入。 例如：1.5 → 1，-1.5 → -2
RoundingMode.HALF_UP	四舍五入，小于等于"五"的向下舍，大于"五"的向上入。 例如：1.5 → 2，1.4 → 1，-1.5 → -2，-1.4 → -1
RoundingMode.HALF_DOWN	四舍五入，小于"五"的向下舍，大于等于"五"的向上入。 例如：1.5 → 1，1.6 → 2，-1.5 → -1，-1.6 → -2
RoundingMode.HALF_EVEN	果舍弃部分左边的数字为奇数，则作 HALF_UP；如果为偶数，则作 HALF_DOWN。 这种方式在统计学中常用，因为它能最大程度地减少舍入操作带来的偏差。 例如：1.5 → 2，2.5 → 2，-1.5 → -2，-2.5 → -2
RoundingMode.UNNECESSARY	断言请求的操作具有精确的结果，因此不需要舍入。 如果结果无法精确表示，则抛出 ArithmeticException

解决方案二：使用 MathContext

```java
public static void main(String[] args) {
    BigDecimal a = new BigDecimal("1.35");
    BigDecimal b = new BigDecimal("3.22");

    BigDecimal divRet = a.divide(b, new MathContext(4,RoundingMode.HALF_UP));
    System.out.println(divRet);
}
//输出结果:
0.4193
```

- new MathContext(4,RoundingMode.HALF_UP)，若不指定的情况下默认是：RoundingMode.HALF_UP模式。

完