BigDecimal用法详情

1. 简介

Java在java.math包中提供了API类BigDecimal，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。在实际应用中，需要对更大或者更小的数进行运算和处理。float和double只能用来做科学计算或者是工程计算，在商业计算中要用java.math.BigDecimal。BigDecimal所创建的是对象，我们不能使用传统的+、-、\*、/ 等算术运算符直接对其对象进行数学运算，而必须调用其相应的方法。方法中的参数也必须是BigDecimal的对象。构造器是类的特殊方法，专门用来创建对象，特别是带有参数的对象。

1. 构造器描述

BigDecimal(int) 创建一个具有参数所指定整数值的对象。

BigDecimal(double) 创建一个具有参数所指定双精度值的对象。

BigDecimal(long) 创建一个具有参数所指定长整数值的对象。

BigDecimal(String) 创建一个具有参数所指定以字符串表示的数值的对象。

注意：此时传入的字符串类型，必须是数字，否则会报错，NumberFormatException

1. 方法

add(BigDecimal) BigDecimal对象中的数值相加，然后返回这个对象。

subtract(BigDecimal) BigDecimal 对象中的值相减，然后返回这个对象

multiply(BigDecimal) BigDecimal 对象中的值相乘，然后返回这个对象

divide(BigDecimal) BigDecimal 对象中的值相除，然后返回这个对象

divide中存在的问题

JAVA中如果用BigDecimal做除法的时候一定要在divide方法中传递第二个参数，使其定义精确到小数点后几位，否则在不整除的情况下，结果是无限循环小数时，就会抛出以上异常。  
 知识点1：

foo.divide(bar, 2, BigDecimal.ROUND\_HALF\_UP);

注意这个divide方法有两个重载的方法，一个是传两个参数的，一个是传三个参数的：

两个参数的方法：

@param divisor value by which this {@code BigDecimal} is to be divided. 传入除数

@param roundingMode rounding mode to apply. 传入round的模式

三个参数的方法：

@param divisor value by which this {@code BigDecimal} is to be divided. 传入除数  
 @param scale scale of the {@code BigDecimal} quotient to be returned. 传入精度  
 @param roundingMode rounding mode to apply. 传入round的模式

知识点2：

BigDecimal 中的 round模式

- BigDecimal.ROUND\_DOWN

截端操作，类似truncate 该模式永远不会增加被操作的数的值

- BigDecimal.ROUND\_UP

在精度最后一位加一个单位 setScale(2,BigDecimal.ROUND\_UP) 1.666 ->1.67

1.011->1.02 1.010->1.01 该模式永远不会减少被操作的数的值

- BigDecimal.ROUND\_CEILING

朝正无穷方向round 如果为正数，行为和round\_up一样，如果为负数，行为和round\_down一样

- BigDecimal.ROUND\_FLOOR

朝负无穷方向round 如果为正数，行为和round\_down一样，如果为负数，行为和round\_up一样

- BigDecimal.ROUND\_HALF\_UP

Rounding mode to round towards {@literal "nearest neighbor"}

\* unless both neighbors are equidistant, in which case round up.

\* Behaves as for {@code ROUND\_UP} if the discarded fraction is

\* &ge; 0.5; otherwise, behaves as for {@code ROUND\_DOWN}. Note

\* that this is the rounding mode that most of us were taught in

\* grade school.

遇到.5的情况时往上近似,例: 1.5 ->;2

- BigDecimal.ROUND\_HALF\_DOWN

/\*\*

\* Rounding mode to round towards {@literal "nearest neighbor"}

\* unless both neighbors are equidistant, in which case round

\* down. Behaves as for {@code ROUND\_UP} if the discarded

\* fraction is {@literal >} 0.5; otherwise, behaves as for

\* {@code ROUND\_DOWN}.

\*/

遇到.5的情况时往下近似,例: 1.5 ->;1 注：1.51->2

- BigDecimal.ROUND\_HALF\_EVEN

/\*\*

\* Rounding mode to round towards the {@literal "nearest neighbor"}

\* unless both neighbors are equidistant, in which case, round

\* towards the even neighbor. Behaves as for

\* {@code ROUND\_HALF\_UP} if the digit to the left of the

\* discarded fraction is odd; behaves as for

\* {@code ROUND\_HALF\_DOWN} if it's even. Note that this is the

\* rounding mode that minimizes cumulative error when applied

\* repeatedly over a sequence of calculations.

\*/

如果舍弃部分左边的数字为奇数，则作 ROUND\_HALF\_UP ；如果它为偶数，则作 ROUND\_HALF\_DOWN

————————————————

版权声明：本文为CSDN博主「tytsp」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/hanlj123/article/details/80581576>

toString() 将BigDecimal对象的数值转换成字符串

注意：

1. char 型数据使用BigDecimal的toString()方法时，进行转换的是其ASCII码值。即’0‘ ---> 48，使用charAt(0) 得到的是 4

2. String 型数据使用BigDecimal的toString()方法时，传入的String类型数据必须是数值，否则报错 ---> java.lang.NumberFormatException

doubleValue() 将BigDecimal 对象中的值以双精度数返回。

注意：

char 与 String 类型数据操作系统同toString类似

floatValue() 将BigDecimal 对象中的值一单精度数返回。

注意：

char 与 String 类型数据操作系统同toString类似

longValue() 将BigDecimal 对象中的值以长整数返回

注意：

char 与 String 类型数据操作系统同toString类似

double 与 float 会出现精度损失问题，即不存在四舍五入

intValue() 将BigDecimal 对象中的值以整数返回

注意：

char 与 String 类型数据操作系统同toString类似

double 与 float 会出现精度损失问题，即不存在四舍五入

1. 格式化及例子

由于NumberFormat类的format()方法可以使用BigDecimal对象作为参数，可以利用BigDecimal对超出16位有效数字的货币值、百分值以及一般数值进行格式化控制。

以利用BigDecimal对货币和百分比格式化为例。首先，创建BigDecimal对象，进行BigDecimal的算术运算后，分别建立对货币和百分比格式化的引用，最后利用BigDecimal对象作为format()方法的参数，输出其格式化的货币值和百分比值。

**public static void** main(String[] args) {  
 NumberFormat currencyInstance = NumberFormat.*getCurrencyInstance*(); // 格式化货币  
 NumberFormat percentInstance = NumberFormat.*getPercentInstance*(); // 格式化百分比  
  
 percentInstance.setMaximumFractionDigits(3); // 设置百分比最大保留位数  
  
 Scanner scanner = **new** Scanner(System.*in*);  
 System.*out*.println("请输入金额");  
 String str = scanner.nextLine();  
 System.*out*.println("请输入利率");  
 String str1 = scanner.nextLine();  
  
 BigDecimal BigDecimal = **new** BigDecimal(str);  
 BigDecimal BigDecimal1 = **new** BigDecimal(str1);  
  
 System.*out*.println("金额：" + currencyInstance.format(BigDecimal.doubleValue()));  
 System.*out*.println("利率：" + percentInstance.format(BigDecimal1.doubleValue()));  
 System.*out*.println("利息：" + BigDecimal.multiply(BigDecimal1));  
  
}

运行结果如下：

请输入金额

200000

请输入利率

0.23

金额：￥200,000.00

利率：23%

利息：46000.00

1. BigDecimal比较

BigDecimal是通过使用compareTo(BigDecimal)来比较的，具体比较情况如下：

public static void main(String[] args) {

BigDecimal a = new BigDecimal("1");

BigDecimal b = new BigDecimal("2");

BigDecimal c = new BigDecimal("1");

int result1 = a.compareTo(b);

int result2 = a.compareTo(c);

int result3 = b.compareTo(a);

System.out.println(result1);

System.out.println(result2);

System.out.println(result3);

}

打印结果分别是：-1、0、1，即左边比右边数大，返回1，相等返回0，比右边小返回-1.

注意：

1. 不能使用equals方法来比较大小
2. 使用BigDecimal的坏处是性能比double和float差，在处理庞大，复杂的运算时尤为明显，因根据实际需求决定使用哪种类型