告警：线上慎用 BigDecimal ， 坑的差点被开了

剽悍一小兔 java小白翻身 2023-05-11 07:51 Posted on 江苏

收录于合集
#Java工作必知必会 6个 >

一、BigDecimal概述

Java在java.math包中提供的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，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。

一般情况下，对于那些不需要准确计算精度的数字，我们可以直接使用Float和Double处理，但是Double.valueOf(String) 和Float.valueOf(String)会丢失精度。所以开发中，如果我们需要精确计算的结果，则必须使用BigDecimal类来操作。

BigDecimal所创建的是对象，故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，而必须调用其相对应的方法。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。

二、BigDecimal常用构造函数

2.1、常用构造函数

- **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

2.2、使用问题分析

使用示例：

```
BigDecimal a =new BigDecimal(0.1);
System.out.println("a values is:"+a);
System.out.println("====================");
BigDecimal b =new BigDecimal("0.1");
System.out.println("b values is:"+b);
```

结果示例：

```
a values is:0.1000000000000000055511151231257827021181583404541015625
====================
b values is:0.1
```

原因分析：

1）参数类型为double的构造方法的结果有一定的不可预知性。有人可能认为在Java中写入newBigDecimal(0.1)所创建的BigDecimal正好等于 0.1（非标度值1，其标度为 1），但是它实际上等于0.1000000000000000055511151231257827021181583404541015625。这是因为0.1无法准确地表示为 double（或者说对于该情况，不能表示为任何有限长度的二进制小数）。这样，传入到构造方法的值不会正好等于 0.1（虽然表面上等于该值）。

2）String 构造方法是完全可预知的：写入 newBigDecimal(“0.1”) 将创建一个BigDecimal，它正好等于预期的 0.1。因此，比较而言， 通常建议优先使用String构造方法。

3）当double必须用作BigDecimal的源时，请注意，此构造方法提供了一个准确转换；它不提供与以下操作相同的结果：先使用Double.toString(double)方法，然后使用BigDecimal(String)构造方法，将double转换为String。要获取该结果，请使用static valueOf(double)方法。

三、BigDecimal常用方法详解

3.1、常用方法

- **add(BigDecimal)**

BigDecimal对象中的值相加，返回BigDecimal对象

- **subtract(BigDecimal)**

BigDecimal对象中的值相减，返回BigDecimal对象

- **multiply(BigDecimal)**

BigDecimal对象中的值相乘，返回BigDecimal对象

- **divide(BigDecimal)**

BigDecimal对象中的值相除，返回BigDecimal对象

- **toString()**

将BigDecimal对象中的值转换成字符串

- **doubleValue()**

将BigDecimal对象中的值转换成双精度数

- **floatValue()**

将BigDecimal对象中的值转换成单精度数

- **longValue()**

将BigDecimal对象中的值转换成长整数

- **intValue()**

将BigDecimal对象中的值转换成整数

3.2、BigDecimal大小比较

java中对BigDecimal比较大小一般用的是bigdemical的compareTo方法

```
int a = bigdemical.compareTo(bigdemical2)
```

返回结果分析：

```
a = -1,表示bigdemical小于bigdemical2;
a = 0,表示bigdemical等于bigdemical2;
a = 1,表示bigdemical大于bigdemical2;
```

举例：a大于等于b

```
new bigdemica(a).compareTo(new bigdemical(b)) >= 0
```

四、BigDecimal格式化

由于NumberFormat类的format()方法可以使用BigDecimal对象作为其参数，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。

```
NumberFormat currency = NumberFormat.getCurrencyInstance(); //建立货币格式化引用
NumberFormat percent = NumberFormat.getPercentInstance();  //建立百分比格式化引用
percent.setMaximumFractionDigits(3); //百分比小数点最多3位

BigDecimal loanAmount = new BigDecimal("15000.48"); //贷款金额
BigDecimal interestRate = new BigDecimal("0.008"); //利率
BigDecimal interest = loanAmount.multiply(interestRate); //相乘

System.out.println("贷款金额:\t" + currency.format(loanAmount));
System.out.println("利率:\t" + percent.format(interestRate));
System.out.println("利息:\t" + currency.format(interest));
```

结果：

```
贷款金额:	￥15,000.48
利率:	0.8%
利息:	￥120.00
```

BigDecimal格式化保留2为小数，不足则补0：

```
public class NumberFormat {

    public static void main(String[] s){
        System.out.println(formatToNumber(new BigDecimal("3.435")));
        System.out.println(formatToNumber(new BigDecimal(0)));
        System.out.println(formatToNumber(new BigDecimal("0.00")));
        System.out.println(formatToNumber(new BigDecimal("0.001")));
        System.out.println(formatToNumber(new BigDecimal("0.006")));
        System.out.println(formatToNumber(new BigDecimal("0.206")));
    }
    /**
     * @desc 1.0~1之间的BigDecimal小数，格式化后失去前面的0,则前面直接加上0。
     * 2.传入的参数等于0，则直接返回字符串"0.00"
     * 3.大于1的小数，直接格式化返回字符串
     * @param obj传入的小数
     * @return
     */
    public static String formatToNumber(BigDecimal obj) {
        DecimalFormat df = new DecimalFormat("#.00");
        if(obj.compareTo(BigDecimal.ZERO)==0) {
            return "0.00";
        }else if(obj.compareTo(BigDecimal.ZERO)>0&&obj.compareTo(new BigDecimal(1))<0){
            return "0"+df.format(obj).toString();
        }else {
            return df.format(obj).toString();
        }
    }
}
```

结果为：

```
3.44
0.00
0.00
0.00
0.01
0.21
```

五、BigDecimal常见异常

5.1、除法的时候出现异常

```
java.lang.ArithmeticException: Non-terminating decimal expansion; no exact representable decimal result
```

原因分析：

通过BigDecimal的divide方法进行除法时当不整除，出现无限循环小数时，就会抛异常：java.lang.ArithmeticException: Non-terminating decimal expansion; no exact representable decimal result.

解决方法：

divide方法设置精确的小数点，如：divide(xxxxx,2)

六、BigDecimal总结

6.1、总结

在需要精确的小数计算时再使用BigDecimal，BigDecimal的性能比double和float差，在处理庞大，复杂的运算时尤为明显。故一般精度的计算没必要使用BigDecimal。 尽量使用参数类型为String的构造函数。

BigDecimal都是不可变的（immutable）的， 在进行每一次四则运算时，都会产生一个新的对象 ，所以在做加减乘除运算时要记得要保存操作后的值。

6.2、工具类推荐

```
package com.vivo.ars.util;
import java.math.BigDecimal;

/**
 * 用于高精确处理常用的数学运算
 */
public class ArithmeticUtils {
    //默认除法运算精度
    private static final int DEF_DIV_SCALE = 10;

    /**
     * 提供精确的加法运算
     *
     * @param v1 被加数
     * @param v2 加数
     * @return 两个参数的和
     */

    public static double add(double v1, double v2) {
        BigDecimal b1 = new BigDecimal(Double.toString(v1));
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
        return b1.add(b2).doubleValue();
    }

    /**
     * 提供精确的加法运算
     *
     * @param v1 被加数
     * @param v2 加数
     * @return 两个参数的和
     */
    public static BigDecimal add(String v1, String v2) {
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
        return b1.add(b2);
    }

    /**
     * 提供精确的加法运算
     *
     * @param v1    被加数
     * @param v2    加数
     * @param scale 保留scale 位小数
     * @return 两个参数的和
     */
    public static String add(String v1, String v2, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException(
                    "The scale must be a positive integer or zero");
        }
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
        return b1.add(b2).setScale(scale, BigDecimal.ROUND_HALF_UP).toString();
    }

    /**
     * 提供精确的减法运算
     *
     * @param v1 被减数
     * @param v2 减数
     * @return 两个参数的差
     */
    public static double sub(double v1, double v2) {
        BigDecimal b1 = new BigDecimal(Double.toString(v1));
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
```

```
        return b1.subtract(b2).doubleValue();
    }

    /**
     * 提供精确的减法运算。
     *
     * @param v1 被减数
     * @param v2 减数
     * @return 两个参数的差
     */
    public static BigDecimal sub(String v1, String v2) {
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
        return b1.subtract(b2);
    }

    /**
     * 提供精确的减法运算
     *
     * @param v1    被减数
     * @param v2    减数
     * @param scale 保留scale 位小数
     * @return 两个参数的差
     */
    public static String sub(String v1, String v2, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException(
                    "The scale must be a positive integer or zero");
        }
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
        return b1.subtract(b2).setScale(scale, BigDecimal.ROUND_HALF_UP).toString();
    }

    /**
     * 提供精确的乘法运算
     *
     * @param v1 被乘数
     * @param v2 乘数
     * @return 两个参数的积
     */
    public static double mul(double v1, double v2) {
        BigDecimal b1 = new BigDecimal(Double.toString(v1));
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
        return b1.multiply(b2).doubleValue();
    }

    /**
     * 提供精确的乘法运算
     *
     * @param v1 被乘数
     * @param v2 乘数
     * @return 两个参数的积
     */
    public static BigDecimal mul(String v1, String v2) {
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
        return b1.multiply(b2);
    }

    /**
     * 提供精确的乘法运算
     *
     * @param v1    被乘数
     * @param v2    乘数
     * @param scale 保留scale 位小数
     * @return 两个参数的积
     */
    public static double mul(double v1, double v2, int scale) {
        BigDecimal b1 = new BigDecimal(Double.toString(v1));
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
        return round(b1.multiply(b2).doubleValue(), scale);
    }

    /**
     * 提供精确的乘法运算
     *
     * @param v1    被乘数
     * @param v2    乘数
     * @param scale 保留scale 位小数
     * @return 两个参数的积
     */
    public static String mul(String v1, String v2, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException(
```

```
                "The scale must be a positive integer or zero");
        }
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
        return b1.multiply(b2).setScale(scale, BigDecimal.ROUND_HALF_UP).toString();
    }

    /**
     * 提供（相对）精确的除法运算，当发生除不尽的情况时，精确到
     * 小数点以后10位，以后的数字四舍五入
     *
     * @param v1 被除数
     * @param v2 除数
     * @return 两个参数的商
     */

    public static double div(double v1, double v2) {
        return div(v1, v2, DEF_DIV_SCALE);
    }

    /**
     * 提供（相对）精确的除法运算。当发生除不尽的情况时，由scale参数指
     * 定精度，以后的数字四舍五入
     *
     * @param v1    被除数
     * @param v2    除数
     * @param scale 表示表示需要精确到小数点以后几位。
     * @return 两个参数的商
     */
    public static double div(double v1, double v2, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException("The scale must be a positive integer or zero");
        }
        BigDecimal b1 = new BigDecimal(Double.toString(v1));
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
        return b1.divide(b2, scale, BigDecimal.ROUND_HALF_UP).doubleValue();
    }

    /**
     * 提供（相对）精确的除法运算。当发生除不尽的情况时，由scale参数指
     * 定精度，以后的数字四舍五入
     *
     * @param v1    被除数
     * @param v2    除数
     * @param scale 表示需要精确到小数点以后几位
     * @return 两个参数的商
     */
    public static String div(String v1, String v2, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException("The scale must be a positive integer or zero");
        }
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v1);
        return b1.divide(b2, scale, BigDecimal.ROUND_HALF_UP).toString();
    }

    /**
     * 提供精确的小数位四舍五入处理
     *
     * @param v     需要四舍五入的数字
     * @param scale 小数点后保留几位
     * @return 四舍五入后的结果
     */
    public static double round(double v, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException("The scale must be a positive integer or zero");
        }
        BigDecimal b = new BigDecimal(Double.toString(v));
        return b.setScale(scale, BigDecimal.ROUND_HALF_UP).doubleValue();
    }

    /**
     * 提供精确的小数位四舍五入处理
     *
     * @param v     需要四舍五入的数字
     * @param scale 小数点后保留几位
     * @return 四舍五入后的结果
     */
    public static String round(String v, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException(
                    "The scale must be a positive integer or zero");
        }
        BigDecimal b = new BigDecimal(v);
        return b.setScale(scale, BigDecimal.ROUND_HALF_UP).toString();
    }
```

```java
    /**
     * 取余数
     *
     * @param v1    被除数
     * @param v2    除数
     * @param scale 小数点后保留几位
     * @return 余数
     */
    public static String remainder(String v1, String v2, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException(
                    "The scale must be a positive integer or zero");
        }
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
        return b1.remainder(b2).setScale(scale, BigDecimal.ROUND_HALF_UP).toString();
    }

    /**
     * 取余数  BigDecimal
     *
     * @param v1    被除数
     * @param v2    除数
     * @param scale 小数点后保留几位
     * @return 余数
     */
    public static BigDecimal remainder(BigDecimal v1, BigDecimal v2, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException(
                    "The scale must be a positive integer or zero");
        }
        return v1.remainder(v2).setScale(scale, BigDecimal.ROUND_HALF_UP);
    }

    /**
     * 比较大小
     *
     * @param v1 被比较数
     * @param v2 比较数
     * @return 如果v1 大于v2 则 返回true 否则false
     */
    public static boolean compare(String v1, String v2) {
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
        int bj = b1.compareTo(b2);
        boolean res;
        if (bj > 0)
            res = true;
        else
            res = false;
        return res;
    }
}
```

收录于合集 #Java工作必知必会6

‹ 上一篇 · Mybatis-plus怎么更新Null字段?

People who liked this content also liked

10 种超好用的 MyBatis 写法
java小白翻身

【SpringBoot系列】vue+SpringBoot实现前后端数据加解密
架构殿堂

lambda之toMap引发的线上故障
java小白翻身