61从零开始学Java61之大数字处理相关的类有哪些?

前言

配套开源项目资料

一. BigInteger类

1. 简介

2. 使用方法

2.1 常用API方法

2.2 基本案例

2.3 类型转换

2.4 其他用法

二. BigDecimal类

1. 简介

2. 使用方法

2.1 常用构造方法

2.2 常用API方法

2.3 divide()除法

2.4 divideAndRemainder()除法

3. 比较两个BigDecimal

4. 配套视频

三. 结语

作者：孙玉昌，昵称【一一哥】，另外【壹壹哥】也是我哦

千锋教育高级教研员、CSDN博客专家、万粉博主、阿里云专家博主、掘金优质作者

前言

我们知道，在现实世界里，实际上数字是有无穷个的，就比如0和1之间，你说有多少个数字？无数个！谁也不知道有多少个吧。但是在计算机中，数字的个数其实是有限的，因为计算机有存储空间的限制，所以实际上无论是整数还是浮点数，都是有最大范围的。比如在Java中，整型的最大范围是64位的long型整数。但是有的小伙伴问了，如果我们使用的整数超过了long型的范围怎么办？此时，我们可以通过软件来模拟一个大整数或大浮点数。

在Java中提供了两个用于大数字运算的类，分别是java.math.BigInteger类 和 java.math.BigDecimal类。这两个类都可以用于高精度计算，BigInteger类是针对整型大数字的处理类，而BigDecimal类是针对大小数的处理类，我们可以用它们来表示任意大小的整数和浮点数。接下来**壹哥**再带大家来学习一下，在Java中如何处理大数字。

----------**前戏已做完，精彩即开始**----------

全文大约【3800】字，不说废话，只讲可以让你学到技术、明白原理的纯干货！本文带有丰富的案例及配图视频，让你更好地理解和运用文中的技术概念，并可以给你带来具有足够启迪的思考.....

配套开源项目资料

Github：

GitHub – SunLtd/LearnJava

Gitee：

一一哥/从零开始学Java

一. BigInteger类

1. 简介

壹哥在之前给大家讲解8种基本类型时就说过，不同的数据类型，有不同的取值范围，我们再通过下表回顾一下：

类型	所占字节(byte)	所占位数(bit)	取值范围
byte	1	8	-2^7 ~ 2^7-1

short	2	16	–2^15 ~2^15–1
int	4	32	–2^31 ~ 2^31–1
long	8	64	–2^63 ~ 2^63–1
char	2字节	16位	0~65535
float	4字节	32位	±3.4E+38
double	8字节	64位	±1.7E+308
boolean	4字节	32位	true\false

从上表中我们可以看到，整型的最大取值范围是–2^63 ~ 2^63–1，浮点型的最大取值范围是±1.7E+308。但是不管这个范围有多大，有些小伙伴就想杠一下，如果我就要存一个比Integer或Long更大的数字，怎么办?

针对这种大整数的需求，我们可以使用BigInteger，它的数字范围比 Integer类型的数字范围要大得多，而且BigInteger支持任意精度的整数。也就是说在运算中，BigInteger类型可以准确地表示任何大小的整数值。BigInteger和Integer、Long一样都是Number的子类，属于不可变类。它自身带有一些可以进行运算的方法，包括基本的加、减、乘、除操作，还有很多较为高级的操作，像求绝对值、相反数、最大公约数及判断是否为质数等，所以BigInteger用起来是比较方便的。

2. 使用方法

2.1 常用API方法

如果我们要使用BigInteger类，首先要创建一个BigInteger对象。BigInteger类提供了多个构造方法，其中最直接的一个是以字符串作为参数的构造方法，即BigInteger(String val)。在创建BigInteger对象之后，我们就可以调用BigInteger类提供的方法，进行各种数学运算了，这些常用的API方法如下：

方法名称	说明
add(BigInteger val)	做加法运算
subtract(BigInteger val)	做减法运算
multiply(BigInteger val)	做乘法运算

divide(BigInteger val)	做除法运算
remainder(BigInteger val)	做取余数运算
divideAndRemainder(BigInteger val)	做除法运算，返回数组的第一个值为商，第二个值为余数
pow(int exponent)	做参数的 exponent 次方运算
negate()	取相反数
shiftLeft(int n)	将数字左移 n 位，如果 n 为负数，则做右移操作
shiftRight(int n)	将数字右移 n 位，如果 n 为负数，则做左移操作
and(BigInteger val)	做与运算
or(BigInteger val)	做或运算
compareTo(BigInteger val)	做数字的比较运算
equals(Object obj)	当参数 obj 是 BigInteger 类型的数字并且数值相等时返回 true, 其他返回 false
min(BigInteger val)	返回较小的数值
max(BigInteger val)	返回较大的数值

2.2 基本案例

我们先来通过一个案例，来验证一下BigInteger中的数字到底有多大。

Java 复制代码

```java
public static void main(String[] args) {
    //创建一个BigInteger对象
    BigInteger bi = new BigInteger("1234567890");
    //计算1234567890的15次方,
    //结果=2358982165591483812094703636914720394831816993851940417596842582341800824911580991261607158852711025590562278956371171634900000000000000000
    System.out.println(bi.pow(15));
}
```

我们会发现，BigInteger可以表示一个非常大的数字，比Integer、Long的范围都要大。

2.3 类型转换

壹哥在上面说过，BigInteger其实是Number的子类，我们知道，Number中定义了几个负责类型转换的方法，比如：

- 转换为byte：byteValue()
- 转换为short：shortValue()
- 转换为int：intValue()
- 转换为long：longValue()
- 转换为float：floatValue()
- 转换为double：doubleValue()

我们利用上述几个方法，就可以把BigInteger转换成基本类型。但是大家要注意，如果BigInteger表示的范围超过了基本类型的范围，在转换时会丢失高位信息，也就是说，结果不一定准确。所以如果我们需要准确地转换成基本类型，可以使用intValueExact()、longValueExact()这样的方法。不过这种方法在转换时如果超出了基本类型的范围，会直接抛出ArithmeticException异常。我们来验证一下吧。

Java | 复制代码

```java
public static void main(String[] args) {
    //BigInteger转基本类型
    BigInteger bi02 = new BigInteger("123456789000");
    //123456789000
    System.out.println("转为int类型="+bi02.intValue());
    System.out.println("转为float类型="+bi02.floatValue());
    System.out.println("转为long类型="+bi02.longValue());

    //将123456789000乘以123456789000，然后将结果转为long类型
    //java.lang.ArithmeticException: BigInteger out of long range
    System.out.println("得到精确结果="+bi02.multiply(bi02).longValueExact())
;
}
```

但是如果BigInteger的值超过了float的最大范围($3.4x10^{38}$)，结果并不会出现ArithmeticException异常，而是会出现Infinity，如下所示：

```java
//计算999999的99次方，并得到该结果的float值
BigInteger bi03 = new BigInteger("999999").pow(99);
float f = bi03.floatValue();
System.out.println("结果="+f);
```

2.4 其他用法

接下来我们再来看看其他的API方法都有哪些作用。

Java | 复制代码

```java
import java.math.BigInteger;
import java.util.Scanner;

/**
 * @author ——哥Sun
 */
public class Demo10 {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("请输入一个整数:");
        // 保存用户输入的数字
        int num = scanner.nextInt();

        // 使用输入的数字创建BigInteger对象
        BigInteger bi = new BigInteger(num + "");
        // 计算大数字加上99的结果
        System.out.println("加法的结果:" + bi.add(new BigInteger("99")));
        // 计算大数字减去25的结果
        System.out.println("减法的结果:" + bi.subtract(new BigInteger("25")));
        // 计算大数字乘以3的结果
        System.out.println("乘法的结果:" + bi.multiply(new BigInteger("3")));
        // 计算大数字除以2的结果
        System.out.println("除法的结果:" + bi.divide(new BigInteger("2")));
        // 计算大数字除以3的商
        System.out.println("取商的结果:" + bi.divideAndRemainder(new BigInteger("3"))[0]);
        // 计算大数字除以3的余数
        System.out.println("取余的结果:" + bi.divideAndRemainder(new BigInteger("3"))[1]);
        // 计算大数字的4次方
        System.out.println("4次方的结果:" + bi.pow(4));
        // 计算大数字的相反数
        System.out.println("取反的结果:" + bi.negate());
    }

}
```

在上述案例中，我们将用户输入的数字作为 BigInteger 对象的参数，然后调用该对象的各种方法，实现了加、减、乘、除等运算，并输出了最终的结果。

二. BigDecimal类

1. 简介

虽然都是用于大数字运算的类，但BigDecimal加入了小数的概念，所以是可以操作小数的。而float 和 double类型，只能用来进行科学计算或工程计算，并不适用于精度要求较高的商业计算(如货币计算)，所以要用到支持任何精度的BigDecimal类。该类中提供了一系列对应的方法，可以用来做超大浮点数的运算，像加、减、乘和除等。在所有运算中，除法运算是最复杂的，因为存在除不尽的情况，需要我们考虑末位小数的处理方式。

2. 使用方法

2.1 常用构造方法

以下是BigDecimal类的常用构造方法：

- BigDecimal(double val)：实例化对象时可以将双精度型转换为BigDecimal类型；
- BigDecimal(String val)：实例化对象时可以将字符串形式转换为BigDecimal类型。

2.2 常用API方法

除了构造方法之外，BigDecimal还提供了一些常用的API方法供我们进行数学运算。这些方法与BigInteger的方法类型，很多方法名称和用法也都与之一致，所以这里**壹哥**就不再一一列出了，接下来我就直接通过一个案例给大家演示这些方法如何使用。

```java
import java.math.BigDecimal;

/**
 * @author ——哥Sun V我领资料：syc_2312119590 各平台都有壹哥的同名博客哦
 */
public class Demo11 {

    public static void main(String[] args) {
        BigDecimal bd = new BigDecimal("1000.05800");
        // 计算大数字加上99的结果
        System.out.println("加法的结果:" + bd.add(new BigDecimal("99")));
        // 计算大数字减去25的结果
        System.out.println("减法的结果:" + bd.subtract(new BigDecimal("25")));
        // 计算大数字乘以1000的结果
        System.out.println("乘法的结果:" + bd.multiply(new BigDecimal(1000)));

        //获取小数的位数，5
        System.out.println(bd.scale());

        //去掉BigDecimal末尾的0，返回一个与原有BigDecimal相等的新对象
        BigDecimal bd2 = bd.stripTrailingZeros();
        System.out.println(bd2.scale());
    }

}
```

在上述代码中，stripTrailingZeros()方法用于去掉BigDecimal末尾的0，并返回一个与原有BigDecimal相等的新对象。而scale()方法用于获取一个数字后面0的个数，如果返回的是负数，比如-2，则表示该数是一个整数，且末尾有2个0。

2.3 divide()除法

BigDecimal进行加、减、乘时，数字的精度不会丢失，但是进行除法运算时，有可能会出现无法除尽的情况，此时必须指定精度以及如何进行截断。BigDecimal给我们提供了divide()和divideAndRemainder()两个方法可以进行除法运算。

其中，divide()方法有3个参数分别表示**除数、商的小数点后的位数和近似值的处理模式**，下表是**壹哥**给大家列出的roundingMode参数支持的处理模式。

模式名称	说明

BigDecimal.ROUND_UP	商的最后一位，如果大于 0，则向前进位，正负数都如此。
BigDecimal.ROUND_DOWN	商的最后一位无论是什么数字都省略
BigDecimal.ROUND_CEILING	商如果是正数，按照 ROUND_UP 模式处理；如果是负数，按照 ROUND_DOWN模式处理
BigDecimal.ROUND_FLOOR	与 ROUND_CELING 模式相反，商如果是正数，按照 ROUND_DOWN 模式处理；如果是负数，按照 ROUND_UP 模式处理
BigDecimal.ROUND_HALF_ DOWN	对商进行五舍六入操作。如果商最后一位小于等于 5，则做舍弃操作，否则对最后一位进行进位操作
BigDecimal.ROUND_HALF_UP	对商进行四舍五入操作。如果商最后一位小于 5，则做舍弃操作，否则对最后一位进行进位操作
BigDecimal.ROUND_HALF_EVEN	如果商的倒数第二位是奇数，则按照 ROUND_HALF_UP 处理；如果是偶数，则按照 ROUND_HALF_DOWN 处理

Java | 复制代码

```java
import java.math.BigDecimal;
import java.math.RoundingMode;

/**
 * @author ——哥Sun
 */
public class Demo12 {

    public static void main(String[] args) {
        BigDecimal d1 = new BigDecimal("123.456");
        BigDecimal d2 = new BigDecimal("123.456789");

        // 会产生ArithmeticException异常，因为除不尽,可以设置RoundingMode，按照指定的方法进行四舍五入或者直接截断：
        //BigDecimal d3 = d1.divide(d2);

        // 保留10位小数并四舍五入
        BigDecimal d4 = d1.divide(d2, 10, RoundingMode.HALF_UP);
        System.out.println("d4="+d4);
        //按指定的位数直接截断，0.xxxx
        BigDecimal d5 = d1.divide(d2, 4, RoundingMode.DOWN);
        System.out.println("d5="+d5);
    }
}
```

2.4 divideAndRemainder()除法

而divideAndRemainder()方法，会返回一个数组，内部包含两个BigDecimal，分别是商和余数，其中商总是整数，余数不会大于除数，所以我们可以利用这个方法来判断两个BigDecimal是否是整数倍数。

Java | 复制代码

```java
import java.math.BigDecimal;
import java.math.RoundingMode;

/**
 * @author ——哥Sun
 */
public class Demo12 {

    public static void main(String[] args) {
        //divideAndRemainder方法，返回一个数组，该数组内部包含了两个BigDecimal，分别是商和余数，其中商总是整数，余数不会大于除数。
        //我们可以利用这个特性来判断两个BigDecimal是否是整数倍数。
        BigDecimal n = new BigDecimal("123.456");
        BigDecimal m = new BigDecimal("0.123");
        BigDecimal[] dr = n.divideAndRemainder(m);
        System.out.println(dr[0]); // 1003
        System.out.println(dr[1]); // 0.087

        if (dr[1].signum() == 0) {
            // n是m的整数倍
            System.out.println("n是m的整数倍");
        }else {
            System.out.println("n不是m的整数倍");
        }
    }
}
```

3. 比较两个BigDecimal

如果我们想比较两个BigDecimal的值是否相等，需要特别注意，请不要使用equals()方法，因为使用该方式进行比较时，不但要求两个BigDecimal的值相等，还要求它们的scale()结果也相等。所以一般是建议使用compareTo()方法来比较，它会根据两个值的大小分别返回负数、正数和0，分别表示小于、大于和等于。如下所示：

```java
import java.math.BigDecimal;

/**
 * @author ——哥Sun
 */
public class Demo13 {

    public static void main(String[] args) {
        BigDecimal d1 = new BigDecimal("123.456");
        BigDecimal d2 = new BigDecimal("123.456000");
        // false,因为scale不同
        System.out.println("d1==d2? "+d1.equals(d2));
        // true,因为d2去除尾部0后scale变为2
        System.out.println("d1==d2? "+d1.equals(d2.stripTrailingZeros()));

         //结果=0，负数表示小于，正数表示大于，0表示等于
        System.out.println("d1==d2? "+d1.compareTo(d2));
    }

}

```

之所以需要使用compareTo()方法来比较两个BigDecimal的值才准确，这是因为一个BigDecimal实际上由一个BigInteger和一个scale组合而成的，其中BigInteger表示一个完整的整数，scale表示小数位数。如下图所示：

```java
public class BigDecimal extends Number implements Comparable<BigDecimal> {
    /**
     * The unscaled value of this BigDecimal, as returned by {@link
     * #unscaledValue}.
     *
     * @serial
     * @see #unscaledValue
     */
    private final BigInteger intVal;

    /**
     * The scale of this BigDecimal, as returned by {@link #scale}.
     *
     * @serial
     * @see #scale
     */
    private final int scale;  // Note: this may have any value, so
```


compareTo()方法内部会对小数位数进行判断，所以更准确，如下图：

```java
@Override
public int compareTo(BigDecimal val) {
    // Quick path for equal scale and non-inflated case.
    if (scale == val.scale) {
        long xs = intCompact;
        long ys = val.intCompact;
        if (xs != INFLATED && ys != INFLATED)
            return xs != ys ? ((xs > ys) ? 1 : -1) : 0;
    }
    int xsign = this.signum();
    int ysign = val.signum();
    if (xsign != ysign)
        return (xsign > ysign) ? 1 : -1;
    if (xsign == 0)
        return 0;
    int cmp = compareMagnitude(val);
    return (xsign > 0) ? cmp : -cmp;
}
```


4. 配套视频

与本节内容配套的视频链接如下：

https://player.bilibili.com/player.html?bvid=BV1FK4y1x7Ny&p=92&page=92

----------正片已结束，来根事后烟----------

三. 结语

至此，**壹哥**就把BigInteger、BigDecimal等大数字类介绍完毕了，最后给大家总结一下今天的重点内容：

> - BigInteger用于表示任意大小的整数；
> - BigInteger是不变类，并且继承自Number；

> - 将BigInteger转换成基本类型时可使用longValueExact()等方法保证结果准确；
> - BigDecimal用于表示精确的小数，常用于财务计算；
> - 比较BigDecimal的值是否相等，必须使用compareTo()而不能使用equals()。

如果你独自学习觉得有很多困难，可以加入**壹哥**的学习互助群，大家一起交流学习。