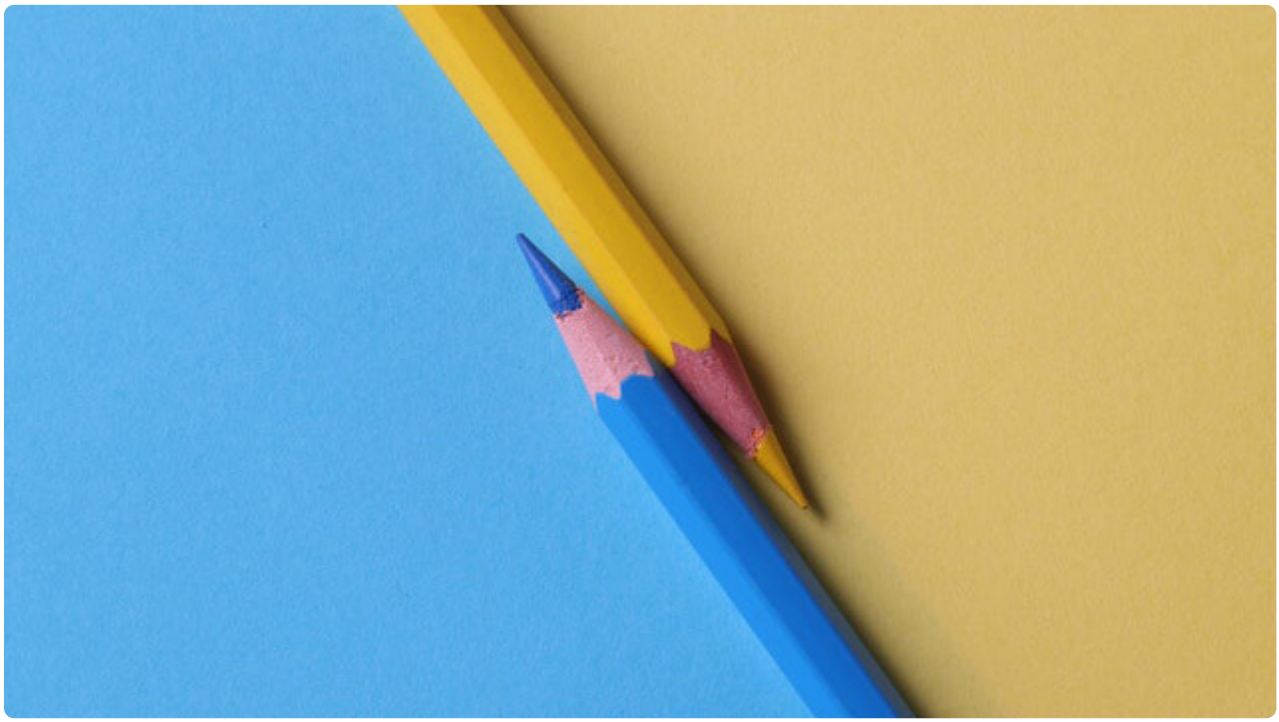
43 专题5：大整数

更新时间：2019-10-16 10:47:33



> 耐心和恒心总会得到报酬的。
>
> ——爱因斯坦

什么是大整数？

在编程中，大整数的概念即是整数数据运算超过了32、64位，计算机里面的简单类型无法满足要求。符合这样条件的整数就被称作为大整数。

背景

内存中一个字节是8位，只能表示2^8=256个数，计算机中无论多复杂的计算，都以一个字节为单位。还有各种扩展数据类型方便我们日常编程使用，例如：

c++数据类型	java数据类型	字节数	范围
byte(char)	byte(char)	1	-2^8 ~ 2^8 - 1
short	short	2	-2^16 ~ 2^16 - 1
int	int	4	-2^32 ~ 2^32 - 1
long long	long	8	-2^64 ~ 2^64 - 1

数据结构和方法

正如 c++ 这种没有大整数支持的语言，超过64位整数，就需要自己封装大数操作来满足我们日常需求；对于python用户或者java用户而言，可以思考大数类的底层实现方式和复杂度。

一个整数的10进制写法，是n位0~9构成的，我们可以用一个只包含0-9的整数数组来表示一个长整数，下面用一个代码例子，来实现一个大整数的类。

内部变量：

- sign：是否正数
- num：一个只包含0-9的正数数组

方法：

- compareTo：对比大小
- toString: 将这个大整数转换成字符串
- valueOf: 将int型数据转换成大整数
- isZero: 是否为0
- opposite: 求相反数
- add: 加法
- minus: 减法

```
import java.util.Arrays;

public class BigInteger implements Comparable<BigInteger> {

    /**
     * 是否整数，1:正数/0，-1:负数
     */
    private int sign;
    /**
     * 大整数数组
     */
    private byte[] num;

    private BigInteger(int sign, byte[] num) {
        this.sign = sign;
        this.num = num;
    }

    public BigInteger add(BigInteger other) {
        //todo
    }

    public BigInteger minus(BigInteger other) {
        //todo
    }

    public boolean isZero() {
        //todo
    }

    public BigInteger opposite() {
        //todo
    }

    public int compareTo(BigInteger other) {
        //todo
    }

    public String toString() {
        //todo
    }
}
```

大整数转成字符串

- 如果是负数，也就是sign=-1的情况，加上'-'

- 从最高位开始一位一位的转成字符串

```
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    if (this.sign < 0) {
        sb.append('-');
    }
    for (int i = this.num.length - 1;i >= 0;i--) {
        sb.append(this.num[i]);
    }
    return sb.toString();
}
```

int转成大整数

- 假设这个待转换的数是num
- 判断num的正负性，如果负数则sign=-1，并将num置为它的绝对值
- 取出最低位(因为我们一开始不知道最高位是哪一位)，将最低位去掉
- 重复上面一步，直到num等于0

```
public static BigInteger valueOf(int data) {
    int sign = 1;
    //int最长就10位
    byte[] num = new byte[10];
    // 真实长度，因为num数组不定长
    int len = 0;
    if (data < 0) {
        sign = -1;
        data = -data;
    }
    while (data != 0) {
        num[len++] = (byte) (data % 10);
        data /= 10;
    }
    // 处理0的情况
    if (len == 0) {
        num[len++] = 0;
    }
    return new BigInteger(sign, Arrays.copyOf(num, len));
}
```

判断是否为0

- 只需要判断num数组长度是否为1，并且这个唯一的元素是否为0

```
public boolean isZero() {
    return this.num.length == 1 && this.num[0] == 0;
}
```

求相反数

- 判断是否为0，如果是0返回本身
- 修改sign

```
public BigInteger opposite() {
    if (isZero()) {
        return this;
    } else {
        return new BigInteger(-sign, num);
    }
}
```

判断大小

- 如果比较的两个数一个正数一个负数，简单
- 如果比较的两个数都是负数，那么把比较它们的绝对值，取反即可
- 如果两个数都是正数，那么有两种情况
    - 位数大的大，位数小的小
    - 位数一样，从高位到低位比较，遇到其中有一位不一样，就做出比较即可

```
@Override
public int compareTo(BigInteger other) {
    if (this.sign > 0 && other.sign > 0) {
        if (this.num.length != other.num.length) {
            return Integer.compare(this.num.length, other.num.length);
        }
        int n = this.num.length;
        for (int i = n - 1;i >= 0;i--) {
            if (this.num[i] != other.num[i]) {
                return Integer.compare(this.num[i], other.num[i]);
            }
        }
        return 0;
    } else if (this.sign < 0 && other.sign < 0) {
        return -this.opposite().compareTo(other.opposite());
    } else if (this.sign < 0 && other.sign > 0) {
        return -1;
    } else {
        return 1;
    }
}
```

加法

- 大整数加法(a+b)，首先判断a和b的符号
- 1、如果a<0, b<0, 则转换成 -((-a)+(-b))
- 2、如果a>0, b<0, 则转换成 a-b
- 3、如果a<0, b>0, 则转换成 b-a
- 4、如果a>0, b>0, 则按位相加

```
public BigInteger add(BigInteger other) {
    if (this.sign < 0 && other.sign < 0) {
        return this.opposite().add(other.opposite()).opposite();
    } else if (this.sign < 0 && other.sign > 0) {
        return other.minus(this.opposite());
    } else if (this.sign > 0 && other.sign < 0) {
        return this.minus(other.opposite());
    } else {
        //两个正整数的加法
        int n = Math.max(this.num.length, other.num.length) + 1;
        //加法结果的位数肯定不会超过最大的那个数的位数+1
        byte[] result = new byte[n];
        result[0] = 0;
        for (int i = 0;i < n - 1;i++) {
            result[i + 1] = 0;
            if (i < this.num.length) {
                result[i] += this.num[i];
            }
            if (i < other.num.length) {
                result[i] += other.num[i];
            }
            //处理进位
            if (result[i] >= 10) {
                result[i + 1]++;
                result[i] -= 10;
            }
        }
        if (result[n - 1] == 0) {
            result = Arrays.copyOf(result, n - 1);
        }
        return new BigInteger(1, result);
    }
}
```

减法

- 大整数减法(a-b)，首先判断a和b的符号
- 1、如果a<0，b<0，则交换下位置，改写成-b-(-a)
- 2、如果a>0，b<0，则转换成-a跟-b相加
- 3、如果a<0, b>0，则改写成-(-a+b)
- 4、如果a>0, b>0，且a<b, 则改写成-(b-a)
- 5、如果a>0, b>0，且a>b，则按位减

```
public BigInteger minus(BigInteger other) {
    if (this.sign < 0 && other.sign < 0) {
        return other.opposite().minus(this.opposite());
    } else if (this.sign < 0 && other.sign > 0) {
        return this.opposite().add(other).opposite();
    } else if (this.sign > 0 && other.sign < 0) {
        return this.add(other.opposite());
    } else {
        int compared = this.compareTo(other);
        if (compared == 0) {
            return BigInteger.valueOf(0);
        } else if (compared < 0) {
            return other.minus(this).opposite();
        } else {
            //两个正整数的减法
            int n = Math.max(this.num.length, other.num.length);
            //减法结果的位数肯定不会超过最大的那个数的位数
            byte[] result = new byte[n];
            int sign = 1;
            result[0] = 0;
            for (int i = 0;i < n;i++) {
                if (i != n - 1) {
                    result[i + 1] = 0;
                }
                if (i < this.num.length) {
                    result[i] += this.num[i];
                }
                if (i < other.num.length) {
                    result[i] -= other.num[i];
                }
                //处理借位
                if (result[i] < 0) {
                    result[i + 1]--;
                    result[i] += 10;
                }
            }
            while (n > 0 && result[n - 1] == 0) {
                n--;
            }
            if (n == 0) {
                //给0留个位置
                n = 1;
            }

            return new BigInteger(1, Arrays.copyOf(result, n));
        }
    }
}
```

小结

今天　这个专题我们学习了大整数的概念、大整数的相关操作、以及大整数的运算。现有的编程语言中像　Python，Java 是有自己封装的大整数类的，但是 C++ 却没有。我们不妨自己试着去实现一个大整数类。可以看下 Python 和 Java 中的整数类源码，学习一下设计思路，然后动手试着自己实现一下。

}