day09【常用API、引用类型小结】

今日内容

- BigInteger类
- BigDecimal类
- Arrays类
- 包装类
- 引用类型的使用场景

教学目标

- [ ] 能够说出BigInteger可以解决的问题
- [ ] 能够说出BigDecimal可以解决的问题
- [ ] 能够使用Arrays类的sort方法
- [ ] 能够使用Arrays类的toString方法
- [ ] 能够说出自动装箱、自动拆箱的概念
- [ ] 能够将基本类型转换为对应的字符串
- [ ] 能够将字符串转换为对应的基本类型
- [ ] 能够在不同的位置使用引用类型

第一章 BigInteger类

1.1 概述

java.math.BigInteger 类，不可变的任意精度的整数。如果运算中，数据的范围超过了long类型后，可以使用 BigInteger类实现，该类的计算整数是不限制长度的。

1.2 构造方法

BigInteger(String value) 将 BigInteger 的十进制字符串表示形式转换为 BigInteger。超过long类型的范围，已经不能称为数字了，因此构造方法中采用字符串的形式来表示超大整数，将超大整数封装成BigInteger对象。

1.3 成员方法

BigInteger类提供了对很大的整数进行加、减、乘、除的方法，注意：都是与另一个BigInteger对象进行运算。

方法声明	描述
add(BigInteger value)	返回其值为 (this + val) 的 BigInteger，超大整数加法运算
subtract(BigInteger value)	返回其值为 (this - val) 的 BigInteger，超大整数减法运算
multiply(BigInteger value)	返回其值为 (this * val) 的 BigInteger，超大整数乘法运算
divide(BigInteger value)	返回其值为 (this / val) 的 BigInteger，超大整数除法运算，除不尽取整数部分

【示例】

```
public static void main(String[] args){
  BigInteger big1 = new BigInteger("832743249325832943264372832084 3");
  BigInteger big2 = new BigInteger("98237473274832382943274328834");
  //加法运算
  BigInteger add = big1.add(big2);
  System.out.println("求和:"+add);
  //减法运算
  BigInteger sub = big1.subtract(big2);
  System.out.println("求差:"+sub);
  //乘法运算
  BigInteger mul = big1.multiply(big2);
  System.out.println("乘积:"+mul);
  //除法运算
  BigInteger div = big1.divide(big2);
  System.out.println("除法:"+div);
}
```

第二章、BigDecimal类

2.1 引入

使用基本类型做浮点数运算精度问题；

看程序说结果：

```
public static void main(String[] args) {
    System.out.println(0.09 + 0.01);//0.09999999999999999
    System.out.println(1.0 - 0.32);//0.6799999999999999
    System.out.println(1.015 * 100);//101.49999999999999
    System.out.println(1.301 / 100);//0.013009999999999999
}
```

对于浮点运算，不要使用基本类型，而使用"BigDecimal类"类型

2.2 概述

相关内容	具体描述
包	java.math 使用时需要导包
类声明	public class BigDecimal extends Number implements Comparable
描述	BigDecimal类提供了算术，缩放操作，舍入，比较，散列和格式转换的操作。提供了更加精准的数据计算方式

2.3 构造方法

构造方法名	描述
BigDecimal(double val)	将double类型的数据封装为BigDecimal对象
BigDecimal(String val)	将 BigDecimal 的字符串表示形式转换为 BigDecimal

注意：推荐使用第二种方式，第一种存在精度问题；

2.4 常用方法

BigDecimal类中使用最多的还是提供的进行四则运算的方法，如下：

方法声明	描述
public BigDecimal add(BigDecimal value)	加法运算
public BigDecimal subtract(BigDecimal value)	减法运算
public BigDecimal multiply(BigDecimal value)	乘法运算
public BigDecimal divide(BigDecimal value)	除法运算

注意：对于divide方法来说，如果除不尽的话，就会出现java.lang.ArithmeticException异常。此时可以使用divide方法的另一个重载方法；

BigDecimal divide(BigDecimal divisor, int scale, int roundingMode): divisor：除数对应的BigDecimal对象； 参数说明：scale:精确的位数；roundingMode取舍模式

小结：Java中小数运算有可能会有精度问题，如果要解决这种精度问题，可以使用BigDecimal

第三章 Arrays类

3.1 Arrays类概述

java.util.Arrays类：该类包含用于操作数组的各种方法（如排序和搜索）

3.2 Arrays类常用方法

- public static void sort(int[] a)：按照数字顺序排列指定的数组
- public static String toString(int[] a)：返回指定数组的内容的字符串表示形式
- 示例代码：

```
public static void main(String[] args) {
    int[] arr = {432, 53, 6, 323, 765, 7, 254, 37, 698, 97, 64, 7};
    //将数组排序
    Arrays.sort(arr);
    //打印数组
    System.out.println(Arrays.toString(arr));
}
```

打印结果：

```
[6, 7, 7, 37, 53, 64, 97, 254, 323, 432, 698, 765]
```

第四章 包装类

4.1 概述

Java提供了两个类型系统，基本类型与引用类型，使用基本类型在于效率，然而很多情况，会创建对象使用，因为对象可以做更多的功能，如果想要我们的基本类型像对象一样操作，就可以使用基本类型对应的包装类，如下：

基本类型	对应的包装类（位于java.lang包中）
byte	Byte
short	Short
int	**Integer**
long	Long
float	Float
double	Double
char	**Character**
boolean	Boolean

4.2 Integer类

- Integer类概述

  包装一个对象中的原始类型 int 的值

- Integer类构造方法及静态方法

方法名	说明
public Integer(int value)	根据 int 值创建 Integer 对象(过时)
public Integer(String s)	根据 String 值创建 Integer 对象(过时)
public static Integer valueOf(int i)	返回表示指定的 int 值的 Integer 实例
public static Integer valueOf(String s)	返回保存指定String值的 Integer 对象

- 示例代码

```
public class IntegerDemo {
    public static void main(String[] args) {
        //public Integer(int value)：根据 int 值创建 Integer 对象(过时)
        Integer i1 = new Integer(100);
        System.out.println(i1);

        //public Integer(String s)：根据 String 值创建 Integer 对象(过时)
        Integer i2 = new Integer("100");
        //Integer i2 = new Integer("abc"); //NumberFormatException
        System.out.println(i2);
        System.out.println("--------");

        //public static Integer valueOf(int i)：返回表示指定的 int 值的 Integer 实例
        Integer i3 = Integer.valueOf(100);
        System.out.println(i3);

        //public static Integer valueOf(String s)：返回保存指定String值的Integer对象
        Integer i4 = Integer.valueOf("100");
        System.out.println(i4);
    }
}
```

4.3 装箱与拆箱

基本类型与对应的包装类对象之间，来回转换的过程称为"装箱"与"拆箱"：

- **装箱**：从基本类型转换为对应的包装类对象。
- **拆箱**：从包装类对象转换为对应的基本类型。

用Integer与 int为例：（看懂代码即可）

基本数值---->包装对象

```
Integer i = new Integer(4);//使用构造函数函数
Integer iii = Integer.valueOf(4);//使用包装类中的valueOf方法
```

包装对象---->基本数值

```
int num = i.intValue();
```

4.4 自动装箱与自动拆箱

由于我们经常要做基本类型与包装类之间的转换，从Java 5（JDK 1.5）开始，基本类型与包装类的装箱、拆箱动作可以自动完成。例如：

```
Integer i = 4;//自动装箱。相当于Integer i = Integer.valueOf(4);
i = i + 5;//等号右边：将i对象转成基本数值(自动拆箱) i.intValue() + 5;
//加法运算完成后，再次装箱，把基本数值转成对象。
```

4.5 基本类型与字符串之间的转换

基本类型转换为String

- 转换方式
- 方式一：直接在数字后加一个空字符串
- 方式二：通过String类静态方法valueOf()
- 示例代码

```
public class IntegerDemo {
    public static void main(String[] args) {
        //int --- String
        int number = 100;
        //方式1
        String s1 = number + "";
        System.out.println(s1);
        //方式2
        //public static String valueOf(int i)
        String s2 = String.valueOf(number);
        System.out.println(s2);
        System.out.println("--------");
    }
}
```

String转换成基本类型

除了Character类之外，其他所有包装类都具有parseXxx静态方法可以将字符串参数转换为对应的基本类型：

- `public static byte parseByte(String s)`：将字符串参数转换为对应的byte基本类型。
- `public static short parseShort(String s)`：将字符串参数转换为对应的short基本类型。
- **`public static int parseInt(String s)`：将字符串参数转换为对应的int基本类型。**
- **`public static long parseLong(String s)`：将字符串参数转换为对应的long基本类型。**
- `public static float parseFloat(String s)`：将字符串参数转换为对应的float基本类型。
- `public static double parseDouble(String s)`：将字符串参数转换为对应的double基本类型。
- `public static boolean parseBoolean(String s)`：将字符串参数转换为对应的boolean基本类型。

代码使用（仅以Integer类的静态方法parseXxx为例）如：

- 转换方式
  - 方式一：先将字符串数字转成Integer，再调用valueOf()方法
  - 方式二：通过Integer静态方法parseInt()进行转换
- 示例代码

```
public class IntegerDemo {
    public static void main(String[] args) {
        //String --- int
        String s = "100";
        //方式1：String --- Integer --- int
        Integer i = Integer.valueOf(s);
        //public int intValue()
        int x = i.intValue();
        System.out.println(x);
        //方式2
        //public static int parseInt(String s)
        int y = Integer.parseInt(s);
        System.out.println(y);
    }
}
```

> 注意:如果字符串参数的内容无法正确转换为对应的基本类型，则会抛出`java.lang.NumberFormatException`异常。

第五章 引用类型使用小结

实际的开发中，引用类型的使用非常重要，也是非常普遍的。我们可以在理解基本类型的使用方式基础上，进一步去掌握引用类型的使用方式。基本类型可以作为成员变量、作为方法的参数、作为方法的返回值，那么当然引用类型也是可以的。在这我们使用两个例子，来学习一下。

5.1 类名作为方法参数和返回值

```
public class Person{
  public void eat(){
    System.out.println("吃饭");
  }
}
public class Test{
  public static void main(String[] args){
        method(new Person());
        Person p = createPerson();
  }

  //引用类型作为方法参数,在前面笔记本案例中我们也使用了接口类型作为方法参数
  pubic static void method(Person p){
       p.eat();
  }

  //引用类型作为返回值
  public static Person createPerson(){
        return new Person();
  }
```

```
}
```

5.2 抽象类作为方法参数和返回值

- 抽象类作为形参：表示可以接收任何此抽象类的"子类对象"作为实参；
- 抽象类作为返回值：表示"此方法可以返回此抽象类的任何子类对象"；

```
/* 定义一个抽象类 */
public abstract class Person{
    public void eat(){
        System.out.println("吃饭");
    }
    public abstract void work();
}
/*定义子类Student*/
public class Student extends Person{
    public void work(){
        System.out.println("学生的工作是学习...");
    }
}
public class Teacher extends Person{
    public void work(){
        System.out.println("老师的工作是教书育人...");
    }
}
/*测试类*/
public class Test{
    public static void main(String[] args){
        //1.调用method1()方法，可以传入一个Student对象，也可以传入一个Teacher对象
        method1(new Student());//OK的
        method1(new Teacher());//OK的

        //2.调用method2()方法，接收的可能是Student对象，也可能是Teacher对象
        //所以，这里使用Person类型接收
        Person p = method2();
        p.work();//如果是Student对象，则调用的是Student的work()；如果是Teacher对象，调
用的是Teacher的work()。
    }
    //-------------------------------------------------------------------------
-----//
    //抽象类Person作为形参--可以接收任何它的子类对象
    public static void method1(Person p){
        p.eat();
        p.work();
    }

    //抽象类Person作为返回值--此方法可以返回一个Student对象，也可以返回一个Teacher对象
    public static Person method2(){
      //  return new Student();//OK的
        return new Teacher();//OK的
    }

}
```

5.3 接口作为方法参数和返回值

- 接口作为方法的形参：【同抽象类】
- 接口作为方法的返回值：【同抽象类】

```
/*定义一个接口*/
public interface USB{
    public void run();
}
/*定义子类*/
public class Keyboard implements USB{
    public void run(){
        System.out.println("使用键盘...");
    }
}
public class Mouse implements USB{
    public void run(){
        System.out.println("使用鼠标...");
    }
}
/*定义测试类*/
public class Test{
    public static void main(String[] args){
        //1.调用method1()方法，需要传入USB的任何子类对象都可以
        method1(new Keyboard());
        method2(new Mouse());

        //2.调用method2()方法，此方法可能返回一个Keyboard对象，也可能返回一个Mouse对象
        USB usb = method2();
        usb.run();
    }
    //接口作为形参
    public static void method1(USB usb){
        usb.run();
    }
    //接口作为返回值
    public static USB method2(){
  //      return new Keyboard();//OK的
        return new Mouse();//OK的
    }
}
```

5.4 类名作为成员变量

我们每个人(Person)都有一个身份证(IDCard)，为了表示这种关系，就需要在Person中定义一个IDCard的成员变量。定义Person类时，代码如下：

```
class Person {
    String name;//姓名
    int age;//年龄
}
```

使用`String`类型表示姓名，`int`类型表示年龄。其实，`String`本身就是引用类型，我们往往忽略了它是引用类型。如果我们继续丰富这个类的定义，给`Person`增加身份证号，身份证签发机关等属性，我们将如何编写呢？这时候就需要编写一个IDCard类了

定义IDCard(身份证)类，添加身份证号，签发地等属性：

```
class IDCard {
    String idNum;//身份证号
    String authority;//签发地

    //getter和setter方法
    //...

    //toString方法
    //...
}
```

修改Person类：

```
public class Person {
    String name;//姓名
    int age;//年龄

    IDCard idCard;//表示自己的身份证信息

    //name和age的getter、setter方法
    //...

    public IDCard getIdCard() {
        return idCard;
    }

    public void setIdCard(IDCard idCard) {
        this.idCard = idCard;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", idCard=" + idCard +
                '}';
    }
}
```

测试类：

```
public class TestDemo {
    public static void main(String[] args) {
        //创建IDCard对象
        IDCard idCard = new IDCard();
        //设置身份证号
        idCard.setIdNum("110113201606066666");
        //设置签发地
        idCard.setAuthority("北京市顺义区公安局");
```

```
        //创建Person对象
        Person p = new Person();
        //设置姓名
        p.setName("小顺子");
        //设置年龄
        p.setAge(2);
        //设置身份证信息
        p.setIdCard(idCard);

        //打印小顺子的信息
        System.out.println(p);
    }
}
输出结果:
Person{name='小顺子', age=2, idCard=IDCard{idNum='110113201606066666',
authority='北京市顺义区公安局'}}
```

> 类作为成员变量时，对它进行赋值的操作，实际上，是赋给它该类的一个对象。同理，接口也是如此，例如我们笔记本案例中使用usb设备。在此我们只是通过小例子，让大家熟识下引用类型的用法，后续在咱们的就业班学习中，这种方式会使用的很多。

5.5 抽象类作为成员变量

- 抽象类作为成员变量——为此成员变量赋值时，可以是任何它的子类对象

```
/*定义抽象类*/
public abstract class Animal{
    public abstract void sleep();
}
/*定义子类*/
public class Cat extends Animal{
    public void sleep(){
        System.out.println("小猫睡觉...");
    }
    public String toString(){
        return "一只可爱的小猫";
    }
}
public class Dog extends Animal{
    public void sleep(){
        System.out.println("小狗睡觉...");
    }
    public String toString(){
        return "一只可爱的小狗";
    }
}
/*定义Student类*/
public class Student{
    private String name;
    private int age;
    private Animal animal;//表示学生有一个动物，可以是Cat，也可以是Dog

    public Student(String name,int age,Animal animal){
        this.name = name;
```

```
        this.age = age;
        this.animal = animal;
    }
    public void setName(String name){
        this.name = name;
    }
    public String getName(){
        return this.name;
    }
    public void setAge(int age){
        this.age = age;
    }
    public int getAge(){
        return this.age;
    }
    public void setAnimal(Animal animal){
        this.animal = animal;
    }
    public Animal getAnimal(){
        return this.animal;
    }

    public String toString(){
        return "Student [name = " + name +
                " , age = " + age +
                " , animal = " + animal +
                "]";
    }
}
/*定义测试类*/
public class Test{
    public static void main(String[] args){
        Student stu = new Student();
        stu.setName("章子怡");
        stu.setAge(19);
    //    stu.setAnimal(new Cat());//表示Student有一只猫
        stu.setAnimal(new Dog());//表示Student有一只狗

        System.out.println(stu);//隐式调用stu.toString()，而stu.toString()中会隐式调用animal的toString().
    }

}
```

5.6 接口作为成员变量

- 接口类型作为成员变量——【同抽象类】

```
/*定义接口*/
public interface Animal{
    public abstract void sleep();
}
/*定义子类*/
public class Cat implements Animal{
```

```
    public void sleep(){
        System.out.println("小猫睡觉...");
    }
    public String toString(){
        return "一只可爱的小猫";
    }
}
public class Dog implements Animal{
    public void sleep(){
        System.out.println("小狗睡觉...");
    }
    public String toString(){
        return "一只可爱的小狗";
    }
}
/*定义Student类*/
public class Student{
    private String name;
    private int age;
    private Animal animal;//表示学生有一个动物，可以是Cat，也可以是Dog

    public Student(String name,int age,Animal animal){
        this.name = name;
        this.age = age;
        this.animal = animal;
    }
    public void setName(String name){
        this.name = name;
    }
    public String getName(){
        return this.name;
    }
    public void setAge(int age){
        this.age = age;
    }
    public int getAge(){
        return this.age;
    }
    public void setAnimal(Animal animal){
        this.animal = animal;
    }
    public Animal getAnimal(){
        return this.animal;
    }

    public String toString(){
        return "Student [name = " + name +
                " , age = " + age +
                " , animal = " + animal +
                "]";
    }
}
/*定义测试类*/
public class Test{
    public static void main(String[] args){
        Student stu = new Student();
        stu.setName("章子怡");
        stu.setAge(19);
```

```
    //    stu.setAnimal(new Cat());//表示Student有一只猫
        stu.setAnimal(new Dog());//表示Student有一只狗

        System.out.println(stu);//隐式调用stu.toString()，而stu.toString()中会隐式调用animal的toString().
    }

}
```