常用类

1.包装类

1.1自动拆装箱

自动装箱不能 自动类型转换

包装类属于引用数据类型在java.lang包下，引用数据类型有共有默认值null;

```
/**
 * 包装类自动装箱
 */
Byte b = 1;
Short s = 10;
Integer i =100;
Long  l = 1000l;
Float f = 10f;
Double d = 10.0;
Character c = 'A';
Boolean bo = true;
```

```
int i1= 10;
//手动装箱
Integer i2 = new Integer(i1);
//自动装箱
Integer inte  = i1;
//静态装箱
Integer i3 = Integer.valueOf(i1);


//拆箱
int j = i2.intValue();

//自动拆箱
int i = i2;

//自动类型转换
double d = 1;
//自动装箱
Double D = d;
//包装类不能自动类型转换
//Double D1= 1;
//可以加上后缀d
Double D2 = 1d;
```

1.2包装类的比较

包装类可以直接使用比较运算符< , > , == , <= , >=

常用的比较方式：比较运算符，compareTo（） ， equals（）

比较运算符说一个 ， == 用在引用数据类型的包装类上的时候，判断的是包装类对象的内存地址。

compareTo（）比较器，用 ”引用.“ 的形式调用括号里传一个要比较的实例（可以是 包装类对象引用，可以是实际数值，可以是手动装箱的包装类对象。）

equals（）也是使用 ”引用.“ 的形式调用，使用equals（）比较的对象类中要重写equals()方法，要不然会使用从Object类中继承的equals（）方法，Object类中的equals（）方法是使用==的方式比较的，比较的是对象的内存地址，（这里的包装类都已经重写了equals（）方法）

```
Integer i = 10;
Integer j = 10;
System.out.println(i < j);
System.out.println(i.compareTo(22));
System.out.println(i.compareTo(j));
System.out.println(i == j);//true
//当数字大于字符串常量池中的数字（-128~127）就会返回 false
System.out.println
(new Integer(128)==new Integer(128));//false

System.out.println(i.equals(j));
```

1.3字符串转成数字

```
/**
 * Integer.parseInt(s1)
    把字符串转换成int（基本数据类型）类型
 * Integer i2 = Integer.valueOf(s2);
    把字符串转换成Integer类型
 *
 * //包装类.parse"包装类"()
 * Double.parseDouble()
 * //包装类.valueOf()构成方法重载，返回调用的包装类类型
 * Double.valueOf()
 */

String s1 = "1";
String s2 = "2";

System.out.println(s1+s2);

int i1 = Integer.parseInt(s1);
Integer i2 = Integer.valueOf(s2);

//自动拆箱
int i3 = Integer.valueOf(s1);
//自动装箱

Integer i4 = Integer.parseInt(s2);
```

```
        System.out.println(i1+i2);

        double d1 = Double.valueOf(s1);
        Double d2 = Double.parseDouble(s1);
```

1.4包装类与基本数据类型的比较

== 与 equals（）

== 比较比较的是两个对象的内存地址是否相等

equals（）比较内存中数据是否相等（包装类均以重写equals）

```
        Integer i1 = new Integer(1);
        Integer i2 = new Integer(1);

        System.out.println(i1);
        System.out.println(i2);
        //== 号比较的是两个对象的内存地址是否相等
        System.out.println(i1 == i2);
        //equals不再去比较内存地址是否相等，转而会去比较内存中数据是否相等
        System.out.println(i1.equals(i2));
```

这里的包装采用了自动装箱机制，因为没有new对象，所以它们存放在堆内存当中。

```
        Integer i3 = 1;
        Integer i4 = 1;

        System.out.println(i3 == i4);
```

i5采用了手动装箱数据存放在堆内存当中，i6采用自动装箱，数据存放在栈内存当中，==比较的是对象内存地址，所以是false 而equals比较的是内存当中的内容，内容都是1，所以返回true

```
        Integer i5 = new Integer(1);
        Integer i6 = 1;
        System.out.println(i5 == i6);
        System.out.println(i5.equals(i6));
```

i7 i8采用自动装箱机制，但是他的数值已经超过了整数常量池当中缓存的数值，所以系统默认new了一个Integer对象出来完成自动装箱，这时候使用==比较内存地址返回false

```
        Integer i7 = 128; // new Integer(128);
        Integer i8 = 128; // new Integer(128);
        System.out.println(i7 == i8);
```

2.String类

2.1String声明并赋值

String字符串类型，可以直接赋值（字符串常量池），也可以通过new对象的方式赋值（堆内存）

```
        String s1 = "中国";
        String s2 = null;
        String s3 = new String("abc");
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
        System.out.println("hello");
```

2.2String类型的比较

str1创建了一个字符串“abc”放到了字符串常量池当中，这时候str2也创建了一个“abc”字符串，不过到字符串常量池的时候发现，常量池中有字符串“abc”他就直接拿去用了，所以str1和str2用的是一个字符串“abc”，所以使用==的方式判断，返回结果为true

equals比较内容返回true

```
        String str1 = "abc";
        String str2 = "abc";
        System.out.println(str1 == str2);
        System.out.println(str3.equals(str4));
```

str3创建的字符串放在字符串常量池，str4的字符串是通过new对象的方式放在了堆内存当中，两个内存都不同地址肯定不一样使用==，返回false

```
        String str3 = "abc";
        String str4 = new String("abc");
        System.out.println(str3 == str4);
```

这里创建了三个字符串对象，三个对象放在字符串常量池当中，问hello == (hel+lo)这里发生了字符串拼接操作hel+lo会再创建一个hello对象==比较的是内存地址所以返回的是false

equals比较内容返回true

```
        String hello = "hello";
        String hel = "hel";
        String lo = "lo";
        System.out.println(hello == (hel+lo));
        System.out.println(hello.equals (hel+lo));
```

2.3intern

```
String s1 = new String("abc");  //s1的空间中
String s2 = "abc";    //字符串常量池中，也就是说字符串常量池中现在有一个abc的内容
String s3 = new String("abc"); //s3的空间中
String s4 = "abc";//字符串常量池中
String s5 = new String("bbc");

System.out.println(s1 == s2);//s1在堆内存当中  s2在字符串常量池（false）
System.out.println(s2 == s3);//s3在堆内存当中  s2在字符串常量池（false）
System.out.println(s1 == s3);//s1和s3是两个不同的实例，因为都new了对象（false）
System.out.println(s2 == s4);//s2和s4都在字符串常量池中，s2创建完s4使用（true）
```

如果字符常量池中有abc,会使用字符串常量池中的abc

```
System.out.println(s1.intern()); //如果字符常量池中有abc,会使用字符串常量池中的abc，也就是变量s2
System.out.println(s5.intern()); //字符串常量中没有bbc，它就会使用自己空间中的bbc,也就是变量s8、
//如果字符常量池中有abc,会使用字符串常量池中的abc
System.out.println(s1.intern() == s1); //S2 == S1
System.out.println(s2 == s1.intern());//s2==s2
System.out.println(s3.intern() == s1.intern());//s2 == s2
```

2.4字符串的常用方法

方法名（实例方法）	返回值类型	功能
trim()	String	去除字符串前后空格
length()	int	获取字符串长度
indexOf()	int	字符首次出现的位置
lastindexOf()	int	字符最后出现的位置
split()	String[]	字符串分割
replace()	String	字符串替换
substring()	String	字符串截取

```
/**
 * String常用方法
 *      str.length()                //字符串长度
 *      str.lastIndexOf("z")        //获取字符最后一次出现下标
 *      str.IndexOf("z")            //获取字符第一次出现下标
 *      string.split("_")           //分割字符串
 *      name.trim();                //去除字符左右空格
 *      message.replace();          //替换字符

 *      String.valueOf(i)           //把整型变成字符串
```

```
 *      m.substring(6, 11);         //截取字符串
 */
String ocean1 = "  zhang hai yang          ";
System.out.println("去除前后字符串空格============《trim》===========");
String ocean = ocean1.trim();

System.out.println("获取字符串长度============《length》===========");
System.out.println(ocean.length());//14

System.out.println("字符最后出现的位置============《lastindexOf》===========");
System.out.println(ocean.lastIndexOf("y"));//10
System.out.println(ocean.lastIndexOf("x"));//-1(如果没有返回-1)

System.out.println("字符首次出现的位置============《indexOf》===========");
System.out.println(ocean.indexOf("h"));//2
System.out.println(ocean.indexOf("x"));//-1(如果没有返回-1)

System.out.println("字符串分割============《split》===========");
String[] s = ocean.split(" ");
for (String str : s){
    System.out.println(str);
}

System.out.println("字符串替换============《replace》===========");
String replace = ocean.replace("hai", "yun");
System.out.println(replace);

System.out.println("字符串截取============《substring》===========");
String substring = ocean.substring(9, 14);
System.out.println(substring);

System.out.println("字符串替换============《replace》===========");
String str = "new String";
String replace1 = str.replace("new ", "niubi");
System.out.println(replace1);
```

2.5字符串效率

每次拼接都要往字符串常量池添加字符串，效率极低

```
String s1 = "abc";
s1 = s1 + "defg";

System.out.println(s1);
System.out.println("----------------------");

//1h = 60min  1min = 60s  1s = 1000ms
 long begin = System.currentTimeMillis();

 String s = "";
System.out.println("开始时间: "+begin);

for(int i = 0 ; i < 1000000 ; i++)
```

```
{
    s += "a";
}

long end = System.currentTimeMillis();
```

2.6StringBuffer

StringBuffer线程安全的，功能，字符串拼接 效率中等比String快

方法名（实例方法）	返回值类型	功能
append()	void	添加字符串
toString()	String	类型转成String类型
delete()	void	删除某个字符
substring()	String	字符串截取
length()	int	获取字符串长度

```
StringBuffer buffer = new StringBuffer("xyz");

buffer.append("bbc");
//"abc"+"bbc"
System.out.println(buffer);
//字符串长度
System.out.println(buffer.length());
buffer.setLength(3);
System.out.println(buffer);
//把一个StringBuffer类型转成String类型
String str = buffer.toString();

buffer.append("bbc");
System.out.println(buffer);

//包前不包后
buffer.delete(1,4);
System.out.println(buffer);

//substring
buffer.append("mnopq");
System.out.println(buffer);

//包前不包后
String msg = buffer.substring(3,6);
System.out.println(msg);

//buffer本身不受影响
```

```
System.out.println(buffer);
```

2.7StringBuilder

StringBuilder线程 非安全的，功能，字符串拼接，效率最高

方法名（实例方法）	返回值类型	功能
append()	void	添加字符串
toString()	String	类型转成String类型
delete()	void	删除某个字符
substring()	String	字符串截取
length()	int	获取字符串长度

```
StringBuilder builder  = new StringBuilder();

builder.append("abc");
builder.append("xyz");

System.out.println(builder);

builder.setLength(3);
System.out.println(builder);

String str = builder.toString();

builder.append("xyz");
builder.delete(1,3);
System.out.println(builder);

String msg = builder.substring(1,3);
System.out.println(msg);
```

3.Date类

获取当前时间

3.1Date类

构造方法	返回值	功能
new Date(Long)	Date	把long类型毫秒数转换成时间

方法名（实例方法）	返回值类型	功能
getTime()	Long	获取到当前时间的毫秒数
toLocaleString()	Date	根据计算机系统时间格式时间
after()	boolean	查看日期是否在，（日期）之前
before()	boolean	查看日期是否在，（日期）之后

```
        //日期对象
        Date date = new Date();
        //会获取到当前时间
        System.out.println(date);
        //JAVA DATE: 1970.1.1    -->  2021.8.5

        //获取到当前时间的毫秒数
        System.out.println(date.getTime());

        //向获取到3天后的日期
        //1天的毫秒数
        Long oneDayTime =Long.valueOf(24*60*60*1000);

        Long day =  date.getTime() + 3*oneDayTime;

        Date d3 = new Date(day);

        System.out.println(d3);

        System.out.println(date.toLocaleString());
        System.out.println(d3.toLocaleString());

        System.out.println(d3.after(date));
        System.out.println(d3.before(date));
```

3.2SimpleDateFormat类

时间格式化

符号	代表含有	备注
yyyy	年	
MM	月	
dd	日	
HH	时	24时
hh	时	12时
mm	分	
ss	秒	
SSS	毫秒	

方法名（实例方法）	返回值类型	功能
format()	String	把Date类型转换为格式化后的字符串
parse()	Date	把一个字符串转换成Date类型

```
String s1 = "yyyy-MM-dd hh:mm:ss";
String s2 = "yyyy/MM/dd hh:mm:ss";
String s3 = "yyyy年MM月dd日 hh时mm分ss秒";
//提供一个格式给这个类
SimpleDateFormat sdf =  new SimpleDateFormat(s3);
//获取到了一个系统时间
Date d = new Date();

//格式化对象方法
String str = sdf.format(d);

System.out.println(str);


System.out.println("------------------------");

String strDate = "1999/9/9 00:00:00";

//字符串转变成为日期类型
SimpleDateFormat sdf2 = new SimpleDateFormat(s2);
```

```
        Date newDate = sdf2.parse(strDate);

        System.out.println(newDate);
```

3.4SimpleDateFormat练习

```
        import java.text.ParseException;
        import java.text.SimpleDateFormat;
        import java.util.Date;
        import java.util.Scanner;

        public class Test03_SimpleDateFormat练习
        {
            public static void main(String[] args) throws ParseException
            {
                Scanner sc = new Scanner(System.in);
                while(true)
                {
                    System.out.println("1、日期类型转成字符类型");
                    System.out.println("2、字符类型转成日期类型");
                    System.out.println("3、获得N天后的日期，用字符串展示");
                    int choose = sc.nextInt();
                    switch (choose)
                    {
                        case 1:
                            Date nowDate = new Date();
                            String strDate = DateHelper.dateToString(nowDate);
                            System.out.println(strDate);

                            break;
                        case 2:
                            System.out.println("请输入日期（yyyy-mm-dd）：");
                            String strDate2 = sc.next();
                            Date nowDate2 = DateHelper.stringToDate(strDate2);
                            System.out.println(nowDate2);

                            break;
                    }

                }
            }
        }
        class DateHelper
        {
            public static String dateToString(Date date)
            {
                SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
                return sdf.format(date);
            }
            public static Date stringToDate(String str) throws ParseException
            {

                SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd");
```

```
            return sdf.parse(str);
        }
    }
```

4.Calendar抽象类

4.1Calendar日历类

方法名（静态方法）	返回值	作用
Calendar.getInstance()	Calender	构建日期对象
Calendar.YEAR	int	获取日历年份
Calendar.DATE	int	获取当月天数
Calendar.DAY_OF_MONTH	int	获取今天是本月第几天

方法名（实例方法）	返回值	作用
get()	int	获取日历内容
set()	void	修改日历日期
setTime(new Date())	void	把当前日期设置进日历

```
//日历
Calendar cal = Calendar.getInstance();
System.out.println(cal);

System.out.println(Calendar.YEAR);

System.out.println(cal.get(Calendar.YEAR));
//通过当前日期，获得本月一共有多少天
System.out.println(cal.getActualMaximum(Calendar.DATE));

//获取到本月中的第几天
System.out.println(cal.get(Calendar.DAY_OF_MONTH));
cal.add(cal.get(Calendar.DAY_OF_MONTH),1);

System.out.println(cal);

//设置日历的日期

cal.set(2020,9,5);
```

```
System.out.println(cal);
System.out.println(cal.get(Calendar.MONTH));


cal.setTime(new Date());
System.out.println(cal);
```

4.2Calender练习

```
Scanner sc = new Scanner(System.in);
//获取日历对象
Calendar cal = Calendar.getInstance();
//格式化日期
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

System.out.println("请输入日期: ");
String str = sc.next();

//把用户输入的日期导入date
Date d = sdf.parse(str);

//把date对象的日期设置进日历
cal.setTime(d);

System.out.println(cal);
//老外是从周日开始算的1  , 中国人是从周一开始算的
//获取是这个日期是周几
int day = cal.get(Calendar.DAY_OF_WEEK);

switch (day)
{
    case 1: case 7:
        System.out.println("今天是休息日");
        break;
    default:
        System.out.println("今天该工作了");
        break;
}
```

5.System类

5.1System常用方法

方法（静态方法）	返回值类型	作用
arraycopy()	void	数组拷贝
gc()	void	垃圾回收
exit(0)	void	退出JVM
currentTimeMillis()	Long	获取1970年至今的毫秒数

```
/**
 * System常用方法
 *          arraycopy()数组拷贝
 *          gc()垃圾回收
 *          exit(0)退出jvm
 *          currentTimeMillis()获取1970年至今的毫秒数
 *
 */
int[] ints = {1,5,6,7,5,22,3};
int[] newInts = new int[ints.length];

long l = System.currentTimeMillis();

//数组拷贝，       老数组   从什么位置   新数组   从第几位      拷贝多少
System.arraycopy(ints,0,newInts,0,ints.length);
for (int i : newInts){
    System.out.print(i);
}

System.gc();

long l1 = System.currentTimeMillis();
System.out.println("程序运行时间-->  " + (l1-l));
```

6.BigDecimal类

需要高精度的数字时建议使用。

6.1BigDecimal类

方法名（实例方法）	返回值	作用
add()	BigDecimal	加
subtract()	BigDecimal	减
multiply()	BigDecimal	乘以
divide()	BigDecimal	除（除数，保留几位，四舍五入）
compareTo()	int	比较器
setScale()	BigDecimal	四舍五入

```
double d = 1.1232323230000000001;
System.out.println(d);
BigDecimal bd = new BigDecimal(d);
System.out.println(bd);

//下面这依旧会造成精度的丢失
//Double d2 = 1.1232323230000000001;
//String s = d2.toString();
String s = "1.1232323230000000001";
String s2 = "12";
BigDecimal bd2 = new BigDecimal(s);
System.out.println(bd2);
BigDecimal bd3 = new BigDecimal(s2);

BigDecimal result = bd2.add(bd3);

System.out.println(result);

result = bd2.subtract(bd3);
System.out.println(result);

result = bd2.multiply(bd3);
System.out.println(result);
//除数     //保留几位   //四舍五入
result = bd2.divide(bd3, 5, BigDecimal.ROUND_HALF_UP);
System.out.println(result);


//比较大小
//bd2 < bd3  ==>-1
//bd2 == bd3  ==>0
//bd2 > bd3  ==>1
System.out.println(bd2.compareTo(bd3));


System.out.println("------------");
```

```
        BigDecimal bd4 = new BigDecimal("1.23232457");
        result = bd4.setScale(3, BigDecimal.ROUND_HALF_UP);

        System.out.println(result);
```

7.Math类

7.1math类

方法名（静态方法）	返回值	作用
Math.random()	double	随机生成一个[0~1)的小数随机数
Math.rint()	double	四舍五入
Math.ceil()	double	向上取整
Math.floor()	double	向下取整

```
        //获取随机
        double r = Math.random();

        System.out.println(r);
        //0 ~ 100的随机数 包含0  包含100   0 - 100.99999
        double r1 = Math.random() * 100 + 1;
        int i = (int)r1;
        System.out.println(i);

        //四色五入
        double r2 = Math.rint(1.645);
        System.out.println(r2);
        //向上取整
        double r3 = Math.ceil(1.145);
        System.out.println(r3);
        //向下取整
        double r4 = Math.floor(1.945);
        System.out.println(r4);
```

8.Random类

8.1Random随机数

方法名（实例方法）	返回值	作用
r.nextInt()	int	返回int取值范围内的一个随机数
r.nextLong()	Long	返回long取值范围内的一个随机数
r.nextDouble()	Double	返回[0,1)取值范围内的一个随机数
nextInt(101)	int	返回[0,101)取值范围内的一个随机数

```
Random r = new Random();
int i = r.nextInt();
System.out.println(i);
int i2 = r.nextInt(11)+10; //0 ~4 但是不包含5;1 ~ 5
System.out.println(i2);

System.out.println("--------------");
long l = r.nextLong();
System.out.println(l);
double d = r.nextDouble(); //0 ~ 0.9999
boolean boo = r.nextBoolean();
System.out.println("--------------");

Random random = new Random();

for(int j = 0 ; j < 100;j++)
{
    int k = random.nextInt(101); // 0 ~ 100
    System.out.println(k);
}
```