1 包装类

我们学习的类都位于 java.base -> java包

`java.lang`：核心包，该包下的类不需要导包

1.1 自动拆装箱

自动装箱不能 自动类型转换

```
//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;
```

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：将8个基本数据类型转成引用数据类型
Byte b = 1;
Short s = 1;
Integer i = 1;
Long l = 1L;
Float f = 1f;
Double d = 1.0;
Boolean boo = true;
Character c = 'c';
```

装箱：将基本数据类型转变成为引用数据类型

```
//手动装箱
Integer in1 = new Integer(1);

//自动装箱
Integer in2 = 1;

//静态装箱：使用静态方法完成手动装箱
Integer in3 = Integer.valueOf(100);

String s = "1001";
//使用手动装箱把String转变为Integer类型
Integer in4 = Integer.valueOf(s);
```

拆箱：将引用数据类型转变成为基本数据类型

```
//自动拆箱:把引用数据类型转成基本数据类型
int i = in3;

//手动拆箱
int j = in3.intValue();
```

1.2 包装类的比较

包装类可以直接使用比较运算符 < , > , == , <= , >= , !=

```
Integer i = 10;
Integer j = 10;

System.out.println(i < j);  //false
System.out.println(i > j);  //false
System.out.println(i == j);  //true
System.out.println(i != j);  //false
```

常用的比较方式：比较运算符、`compareTo()`、`equals()`

1 ）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。

```
Integer i = 10;
Integer j = 10;
System.out.println(i < j);  //false
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
```

当我们保存的数值超过了127，那么系统会自动帮我们去创建对象（new 对象），因为在保存 -128 ~ 127之间的数字时，Java觉得老创建对象太浪费资源，所以将这些数字存放在栈内存中，方便开发人员使用，一旦超过了127,那么系统会自动创建对象，将数据放在堆内存中，一旦使用了new，那么 == 比较的内容就一定不一样了。

2 ） `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。 ）

```
//compareTo 方法，是一个比较器
System.out.println(i1.compareTo(i2));  //0     i1 == i2
System.out.println(i3.compareTo(i4));  //-1    i3 < i4
System.out.println(i5.compareTo(i6));  //1     i5 > i6
```

比较器用来比较"引用"和"实例"的大小：

1. 引用 > 实例，返回 1；
2. 引用 < 实例，返回 -1；
3. 引用 == 实例，返回 0；

3） `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 o = new Integer(1);
Integer p = 1;
System.out.println(o == p);  //false
System.out.println(o.equals(p));  //true
```

`equals()` 方法，比较的是数值是否一样。

1.3 字符串转成数字

```
String s1 = "1";
String s2 = "2";
System.out.println(s1 + s2);  //12

//把字符串转换成int（基本数据类型）类型
int i1 = Integer.parseInt(s1);  //1
//把字符串转换成Integer类型
Integer i2 = Integer.valueOf(s2);  //2
System.out.println(i1 + i2);  //3

//自动拆箱：包装类.valueOf()构成方法重载，返回调用的包装类类型
int i3 = Integer.valueOf(s1);  //1
//自动装箱：包装类.parse"包装类"()，返回"包装类"类型
Integer i4 = Integer.parseInt(s2);  //2

double d1 = Double.valueOf(s1);  //1.0
Double d2 = Double.parseDouble(s1);  //1.0
```

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);  //false
//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));  //true
```

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);  //true
```

i5 i6 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 i5 = 128;  // new Integer(128);
Integer i6 = 128;  // new Integer(128);

System.out.println(i5 == i6);  //false
```

2 String类

2.1 String类声明并赋值

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";  //字符串常量池
String s2 = null;
String s3 = new String("abc");  //堆内存
```

2.2 String类型的比较

```
String str1 = "abc";
String str2 = "abc";
System.out.println(str1 == str2);  //true
System.out.println(str1.equals(str2));  //true
```

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

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 hello = "hello";
String hel = "hel";
String lo = "lo";

System.out.println(hello == (hel+lo));  //false
System.out.println(hello.equals(hel+lo));  //true
```

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。

2.3 intern方法

intern() 方法返回一个在字符串常量池中的字符串。

```
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）
```

在使用`String 变量名 = "字符串";`时，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);  //false
System.out.println(s2 == s1.intern());  //true
System.out.println(s3.intern() == s1.intern());  //true
```

s1.intern() 用的是字符串常量池中的字符串，s1是堆空间的字符串。

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

```
//去除前后空格
String s1 = "    hello say hi   ";
System.out.println(s1.trim());  //hello say hi

//获取字符串长度  从1开始
String s2 = "  hello world ";
System.out.println(s1.length());   //19

//获取字符首次出现的位置  从0开始  没有返回-1
String s3 = "acksoeufhdioshdufi";
System.out.println(s3.indexOf("f"));  //7

//获取字符最后出现的位置  从0开始  没有返回-1
String s4 = "spadpwaduiiaheif";  从0开始
System.out.println(s4.lastIndexOf("a"));  //11

//字符串分割
String s5 = "sd ed fg ht hj yh";
String[] s6 = s5.split(" ");
System.out.println(Arrays.toString(s6));

//字符串替换
String s7 = "sa ok sw_sd lk,sd,oi_sw sd";
String s8 = s7.replace("_", " ").replace(",", " ");
System.out.println(s8);  //sa ok sw sd lk sd oi sw sd
String[] s9 = s8.split(" ");
System.out.println(Arrays.toString(s9));

//字符串截取  开始下标 ~ 结束下标  从0开始  到加1结束  包左不包右
String s10 = "ppppooooiiiiuuuujjjj";
int begIndex = s10.indexOf("i");
int endIndex = s10.lastIndexOf("i") + 1;
String s11 = s10.substring(begIndex, endIndex);
System.out.println(s11);  //iiii
```

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 s = "a";

long begin = System.currentTimeMillis();

for (int i = 0; i < 1000000; i++) {
    s = s + "a";
}

long end = System.currentTimeMillis();

System.out.println(end - begin);  //73945 毫秒
```

2.6 StringBuffer

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");

//"abc"+"bbc"
buffer.append("bbc");
System.out.println(buffer);  //xyzbbc

//字符串长度
System.out.println(buffer.length());  //6
//修改字符串长度为3
buffer.setLength(3);
System.out.println(buffer);  //xyz

//把一个StringBuffer类型转成String类型
String str = buffer.toString();

buffer.append("bbc");

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

```
//包前不包后  从0开始
buffer.delete(1, 4);
System.out.println(buffer);  //xbc

//substring
buffer.append("mnopq");
System.out.println(buffer);  //xbcmnopq
//包前不包后  从0开始
String msg = buffer.substring(3, 6);
System.out.println(msg);  //mno
//buffer本身不受影响
System.out.println(buffer);  //xbcmnopq
```

```
StringBuffer s = new StringBuffer("a");

long begin = System.currentTimeMillis();

for (int i = 0; i < 100000000; i++) {
    s.append("a");
}

long end = System.currentTimeMillis();

System.out.println(end - begin);  //625 毫秒
```

2.7 StringBuilder

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);  //abcxyz


builder.setLength(3);
```

```
System.out.println(builder);  //abc

String str = builder.toString();

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

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

```
StringBuilder s = new StringBuilder("a");

long begin = System.currentTimeMillis();

for (int i = 0; i < 100000000; i++) {
    s.append("a");
}

long end = System.currentTimeMillis();

System.out.println(end - begin);  //438 毫秒
```

3 Date日期类

获取当前时间。

3.1 Date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
new Date()	Date	获取当前时间

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

```
Long oneDayTime = 1 * 24 * 60 * 60 * 1000l;
Date d1 = new Date();
```

```
//英文格式
System.out.println(d1);
//格式化 输出
System.out.println(d1.toLocaleString());

//获取当前时间的毫秒数
Long d2 = d1.getTime();
d2 += oneDayTime;

//将Long类型毫秒数转成Date类型
Date d3 = new Date(d2);
System.out.println(d3.toLocaleString());

//d1是否在d3之后
System.out.println(d1.after(d3));  //false
//d1是否在d3之前
System.out.println(d1.before(d3));  //true
//d1是否等于d3
System.out.println(d1.equals(d3));  //false
```

3.2 SimpleDateFormat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 SSS";
String s2 = "yyyy/MM/dd hh:mm:ss";
```

```
String s3 = "yyyy年MM月dd日 HH:mm:ss";
SimpleDateFormat sdf1 = new SimpleDateFormat(s1);
SimpleDateFormat sdf2 = new SimpleDateFormat(s2);
SimpleDateFormat sdf3 = new SimpleDateFormat(s3);
//实例化日期对象  英文格式日期类型
Date date = new Date();
System.out.println(date); //Thu Jul 07 09:18:43 GMT+08:00 2022
//类型格式化
String sd1 = sdf1.format(date);
System.out.println(sd1); //2022-07-07 09:18:43 540

String sd2 = sdf2.format(date);
System.out.println(sd2); //2022/07/07 09:18:43

String sd3 = sdf3.format(date);
System.out.println(sd3); //2022年07月07日 09:18:43
```

```
String str = "1999-9-9";
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

Date d = sdf.parse(str);
System.out.println(d); //Thu Sep 09 00:00:00 GMT+08:00 1999
```

在使用parse() 方法需要添加异常到方法签名（throws ParseException），或者使用try/catch环绕。

4 Calendar日历类

日历类

方法名（静态方法）	返回值	作用
Calendar.getInstance()	Calendar	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

```
Calendar cl = Calendar.getInstance();
```

```
System.out.println(cl);
//...DAY_OF_MONTH=7,DAY_OF_YEAR=188,DAY_OF_WEEK=5,DAY_OF_WEEK_IN_MONTH=1...
System.out.println("获取年：" + cl.get(Calendar.YEAR)); //获取年：2022
System.out.println(cl.get(Calendar.MONTH)); //6
System.out.println(cl.get(1)); //2022
System.out.println(cl.get(Calendar.DATE)); //7
//获取当前月一共有多少天
System.out.println(cl.getActualMaximum(Calendar.DATE)); //31

//添加本周的第几天+1
cl.add(cl.get(Calendar.MONTH), 1);
System.out.println(cl);
//...DAY_OF_MONTH=8,DAY_OF_YEAR=189,DAY_OF_WEEK=6,DAY_OF_WEEK_IN_MONTH=2...

cl.set(2023, 7, 4);
System.out.println(cl);

System.out.println(cl.get(Calendar.YEAR) + "年" + cl.get(Calendar.MONTH) + "月" +
cl.get(Calendar.DATE) + "日"); //2023年7月4日
```

```
Calendar cl = Calendar.getInstance();
System.out.println(cl);  //2022.7.7 9:41:53 113
/*
java.util.GregorianCalendar[time=1657158113113,areFieldsSet=true,areAllFieldsSet=true,lenient=tr
ue,zone=sun.util.calendar.ZoneInfo[id="GMT+08:00",offset=28800000,dstSavings=0,useDaylight=false
,transitions=0,lastRule=null],firstDayOfWeek=1,minimalDaysInFirstWeek=1,ERA=1,YEAR=2022,MONTH=6,
WEEK_OF_YEAR=28,WEEK_OF_MONTH=2,DAY_OF_MONTH=7,DAY_OF_YEAR=188,DAY_OF_WEEK=5,DAY_OF_WEEK_IN_MONT
H=1,AM_PM=0,HOUR=9,HOUR_OF_DAY=9,MINUTE=41,SECOND=53,MILLISECOND=113,ZONE_OFFSET=28800000,DST_OF
FSET=0]
*/
```

5 System类

方法名（静态方法）	返回值	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

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

int[] oldArray = {1, 2, 3, 4, 5, 6};

int[] newArray = new int[oldArray.length];
```

```
//数组拷贝
System.arraycopy(oldArray, 0, newArray, 0, oldArray.length);
System.out.println(Arrays.toString(newArray)); //[1, 2, 3, 4, 5, 6]
for (int i : newArray) {
    System.out.print(i + " ");
}
System.out.println();

//垃圾回收
System.gc();

//获取1970年至今的毫秒数
long end = System.currentTimeMillis();
System.out.println(end); //1657158943496
System.out.println("程序运行时间--> " + (end - begin)); //15

//退出JVM了
System.exit(10); //Process finished with exit code 10
```

6 BigDecimal大数据类

需要高精度的数字时建议使用。

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

```
//精度丢失
double d1 = 1.0;
double d2 =0.9;
System.out.println(d1 - d2); //0.09999999999999998

double d = 1.1232323230000000001;
System.out.println(d); //1.123232323

BigDecimal bd = new BigDecimal(d);
System.out.println(bd); //1.12323232300000008798690487310523167252540588378906250


//数字转成为字符串
```

```
String s = d + "";
Double dd = 1.1232323230000000001;
String ss = dd.toString();
System.out.println(s); //1.123232323
System.out.println(ss); //1.123232323

//防止精度丢失 在接受数据的时候，直接用字符串保存
Scanner sc = new Scanner(System.in);
System.out.println("请输入高精度数字：");
String msg = sc.next();
System.out.println(msg); //1.1232323230000000001

BigDecimal bd2 = new BigDecimal(msg);
BigDecimal bd3 = new BigDecimal(msg);
System.out.println(bd2); //1.1232323230000000001
//加法
System.out.println(bd2.add(bd3)); //2.2464646460000000002
//减法
System.out.println(bd2.subtract(bd3)); //0E-19
//乘法
System.out.println(bd2.multiply(bd3)); //1.261650851431976329224646464600000000001
//除法
BigDecimal result = bd2.divide(bd3,3,BigDecimal.ROUND_HALF_UP);
System.out.println(result); //1.000

/*
    比较器
    bd2 > bd3  返回1
    bd2 < bd3  返回-1
    bd2 = bd3  返回0
 */
System.out.println(bd2.compareTo(bd3)); //0
```

7 Math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.round()	long	四舍五入
Math.rint()	double	四舍五入
Math.ceil()	double	向上取整
Math.floor()	double	向下取整

```
//0 ~ 1  不包含1
double d = Math.random();
System.out.println(d);
```

```
//1~100
int d2 = (int)(Math.random()*100) + 1;
System.out.println(d2);

double number = 3.8;
//四舍五入
System.out.println(Math.rint(number)); //4.0
System.out.println(Math.round(number)); //4
//向上取整
System.out.println(Math.ceil(number)); //4.0
//向下取整
System.out.println(Math.floor(number)); //3.0
```

8 Random类

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
r.nextInt(101)	int	返回[0,101)取值范围内的一个随机数

```
Random r = new Random();

//整个int取值范围的随机数
System.out.println(r.nextInt()); //-300424520
//整个long取值范围的随机数
System.out.println(r.nextLong()); //2659846247294413941
//[0,1)取值范围内的一个随机数
System.out.println(r.nextDouble()); //0.9317026035282631
//[1,101)取值范围内的一个随机数
int i = r.nextInt(101) + 1;
System.out.println(i); //29
```