福建師範大學
FUJIAN NORMAL UNIVERSITY

Java语言

学　　期：2025-2026-1

教　　师：赵珊珊

第7章Java常用系统类

第7章Java常用系统类

- 7.1基本类型包装器
- 7.2字符串类
- 7.3System类与Runtime类
- 7.4Math类与Random类
- 7.5日期类

7.1基本类型包装器

- JDK中提供了一系列的包装类可以将基本数据类型的值包装为引用数据类型的对象。
- Java中的8种基本数据类型都有与之对应的包装类

基本数据类型	包装类	基本数据类型	包装类
int	Integer	byte	Byte
char	Character	long	Long
float	Float	short	Short
double	Double	boolean	Boolean

7.1基本类型包装器

- 包装类和基本数据类型进行转换时要涉及两个概念——装箱和拆箱。
- 装箱是指将基本数据类型的值转为引用数据类型的对象
- 拆箱是指将引用数据类型的对象转为基本数据类型。

- JDK提供了自动装箱和拆箱机制，是指基本类型值与包装类的对象相互自动转换，在变量赋值或方法调用等情况时，使用上更加简单直接，从而提高了开发效率。

7.1基本类型包装器

自动拆箱和装箱是 Java 5 引入的语法，它们让基本数据类型和对应的包装类之间的转换自动完成，让代码更简洁。

JDK1.5之前

```
// 手动装箱
int num = 10;
Integer integerObj = Integer.valueOf(num); // 或 new Integer(num)
// 手动拆箱
Integer anotherIntegerObj = Integer.valueOf(20);
int primitiveNum = anotherIntegerObj.intValue();
```

JDK1.5及以后

```
// 自动装箱：基本类型 int 直接赋给 Integer
Integer integerObj = 10; // 编译器自动改为 Integer.valueOf(10)
// 自动拆箱：Integer 对象直接赋给 int
int primitiveNum = integerObj; // 编译器自动改为 integerObj.intValue()
```

由于集合只能存储对象，自动拆装箱在操作集合（特别是泛型集合）时的体验差异最为明显。

无自动封装箱

```
import java.util.ArrayList;
public class Main {
    public static void main(String[] args) {
        int[] intArray = {1, 2, 3, 4, 5}; // 基本类型数组
        ArrayList integerList = new ArrayList(); // 无法使用泛型，默认为Object
        // 1. 手动装箱：将基本类型 int 转换为 Integer 对象才能加入集合
        for (int i = 0; i < intArray.length; i++) {
            // 必须显式地创建 Integer 对象
            Integer integerObj = Integer.valueOf(intArray[i]);
            integerList.add(integerObj);
        }
        int sum = 0;
        // 2. 手动拆箱：从集合中取出 Integer 对象后，再转换为基本类型 int 进行求和
        for (int i = 0; i < integerList.size(); i++) {
            // 取出的是 Object 类型，需要先强制转换为 Integer
            Integer integerObj = (Integer) integerList.get(i);
            // 再调用 intValue() 方法得到基本类型 int
            sum += integerObj.intValue();
        }
        System.out.println("Sum is: " + sum);
    }
}
```

自动封装箱

```
import java.util.ArrayList;
import java.util.List;
public class Main {
    public static void main(String[] args) {
        int[] intArray = {1, 2, 3, 4, 5}; // 基本类型数组
        List<Integer> integerList = new ArrayList<>(); // 有了泛型，指定存储Integer
        // 1. 自动装箱：编译器自动将 int 转换为 Integer，然后加入集合
        for (int num : intArray) {
            integerList.add(num); // 等价于 integerList.add(Integer.valueOf(num));
        }
        int sum = 0;
        // 2. 自动拆箱：编译器自动将 Integer 对象转换为 int 进行求和
        for (int num : integerList) {
            sum += num; // 等价于 sum += num.intValue();
        }
        System.out.println("Sum is: " + sum);
    }
}
```

7.2字符串类

- 7.2.1 String类
- String类表示不可变的字符串，一旦String类被创建，该对象中的字符序列将不可改变，直到这个对象被销毁。
- 在Java中，字符串被大量使用，为了避免每次都创建相同的字符串对象及内存分配，JVM内部对字符串对象的创建做了一些优化，用一块内存区域专门来存储字符串常量，该区域被称为常量池。

7.2字符串类

- 7.2.1 String类
- 直接赋值初始化

```
public class StringCst {
    private String name1 = "NorthPark";
    private String name2 = "NorthPark";
    private String nameBegin="North";
    private String nameEnd = "Park";
    private String nameAdd = nameBegin+nameEnd;
    StringCst() {
        if(name1==name2) 比较==、equals、compareTo
            System.out.println("直接赋值name1和name2==的结果相等");
        else
            System.out.println("直接赋值name1和name2==的结果不相等");
        if(name1.equals(name2))
            System.out.println("直接赋值name1和name2 equlas的结果相等");
        else
            System.out.println("直接赋值name1和name2 equlas的结果不相等");
        if(name1.compareTo(name2)==0)
            System.out.println("直接赋值name1和name2 compareTo的结果相等");
        else
            System.out.println("直接赋值name1和name2 compareTo的结果不相等");
        if(name2==nameAdd)
            System.out.println("name2:NorthPark和nameAdd:North+Park的结果相等");
        else
            System.out.println("name2:NorthPark和nameAdd:North+Park的结果不相等");
    }
    public static void main(String[] args) {
        new StringCst();
    }
}
```

```
直接赋值name1和name2==的结果相等
直接赋值name1和name2 equlas的结果相等
直接赋值name1和name2 compareTo的结果相等
name2:NorthPark和nameAdd:North+Park的结果不相等
```

构造方法初始化

```
public class StringCst {
    private String name1 = "NorthPark";
    private String name2 = new String("NorthPark");
    private String name3 = new String("NorthPark");
    StringCst() {
        if(name1==name2)
            System.out.println("构造方法name1和name2==的结果相等");
        else
            System.out.println("构造方法name1和name2==的结果不相等");
        if(name1==name3)
            System.out.println("构造方法name1和name3==的结果相等");
        else
            System.out.println("构造方法name1和name3==的结果不相等");
        if(name2==name3)
            System.out.println("构造方法name2和name3==的结果相等");
        else
            System.out.println("构造方法name2和name3==的结果不相等");
    }
    public static void main(String[] args) {
        new StringCst();
    }
}
```

new开辟新内存，地址不同

```
构造方法name1和name2==的结果不相等
构造方法name1和name3==的结果不相等
构造方法name2和name3==的结果不相等
```

7.2字符串类

- 7.2.1 String类
- 常见方法

P127 表7-5

charAt(int index)　int compareTo(String anotherString)　Boolean equals(Object anO)
getChars(int strbegin,int strend, char[] data, int offset)
int length()　int indexOf(int ch)…
String toLowerChase()/toUpperCase()　String toString()　String trim()
String substring(int strbegin, int strend)

7.2字符串类

- 7.2.2StringBuffer类

StringBuffer**线程安全**，适合多线程环境。

- 字符串缓冲区StringBuffer支持可变的字符串，它的对象是可以扩充和修改的。在字符串的内容会不断修改的时候使用StringBuffer比较合适。

```
public class StringCst {
    public static void main(String[] args) {
        StringBuffer sb1=new StringBuffer();
        sb1.append("Hell");//添加string
        sb1.append('o');//添加char
        StringBuffer sb2=new StringBuffer();
        sb2.append("\tworld");
        sb1.append(sb2);//添加stringbuffer对象
        System.out.println("sb1+sb2="+sb1);
        sb1.delete(5, 6);//删除字符或字符串
        System.out.println("删除第6个字符后："+sb1);
        sb1.insert(5,"—");//插入字符或字符串
        System.out.println("插入—后："+sb1);
        System.out.println("反转后："+sb1.reverse());//反转
    }
}
```

```
sb1+sb2=Hello    world
删除第6个字符后：Helloworld
插入—后：Hello—world
反转后：dlrow—olleH
```

7.3System类与Runtime类

- Java程序在不同操作系统上运行时，可能需要取得平台相关的属性，或者调用平台命令来完成特定功能。Java提供了System类和Runtime类与程序的运行平台进行交互。
  - System类

System.exit()会立即终止当前正在运行的 Java 虚拟机，导致所有线程停止

currentTimeMillis()- 获取当前时间戳（毫秒）

System.gc()仅仅是建议 JVM 进行垃圾回收，并不能保证垃圾回收一定会立即发生。最好让 JVM 自行管理内存，避免显式调用该方法

System.getProperties()返回一个包含所有系统属性的 Properties对象，而 System.getProperty(String key)则用于获取指定键对应的属性

```
public class StringCst {
    public static void main(String[] args) {
        System.out.println("当前系统版本为："+System.getProperty("os.name")+"\t"
        +System.getProperty("os.version")+"\t"+System.getProperty("os.arch"));
        System.out.println("当前系统工作目录为："+System.getProperty("user.dir"));
        System.out.println("当前系统信息为："+System.getProperties());
    }
}
```

```
当前系统版本为：Windows 10       10.0    amd64
当前系统工作目录为：D:\javaprogram\FJNUObject
当前系统信息为：{java.specification.version=17, sun.cpu.isalist=ar
, java.vm.specification.vendor=Oracle Corporation, java.specifica
```

```
public static void main(String[] args) throws Exception{
    System.out.println("系统启动...");
    long start = System.currentTimeMillis();
    Thread.sleep(1000);
    long end = System.currentTimeMillis();
    System.out.println("系统睡眠了"+(end-start)/1000+"秒");
}
```

```
系统启动...
系统睡眠了1秒
```

7.3System类与Runtime类

- Runtime类

```
public class StringCst {
    public static void main(String[] args) throws Exception{
        Runtime runtime = Runtime.getRuntime();
        System.out.println("可用处理器数量: " + runtime.availableProcessors());
        System.out.println("空闲内存数: " + runtime.freeMemory()/1024/1024+"MB");
        System.out.println("总内存数: " + runtime.totalMemory()/1024/1024+"MB");
        System.out.println("可用最大内存数:  " + runtime.maxMemory()/1024/1024+"MB");
        runtime.exec("notepad.exe");
        runtime.exec("C:\\Program Files\\Microsoft Office\\root\\Office16\\WINWORD.EXE");
    }
}
```

```
可用处理器数量: 8
空闲内存数: 252MB
总内存数: 254MB
可用最大内存数:  4050MB
```

7.4Math类与Random类

- 7.4.1Math类
  - java.lang.Math类包含用于几何学、三角学及几种一般用途的浮点类方法，来执行很多数学问题。

类别	方法名	作用描述	示例
常用常量	Math.PI	圆周率 π (约 3.141592653589793)	double area = Math.PI * r * r;
	Math.E	自然常数 e (约 2.718281828459045)	double exp = Math.pow(Math.E, 2);
基本运算	Math.abs(x)	返回 x 的绝对值 (支持 int, long, float, double)	Math.abs(-5); // 5Math.abs(-3.14); // 3.14
	Math.max(a, b)	返回 a 和 b 中的最大值 (支持多种数值类型)	Math.max(10, 20); // 20
	Math.min(a, b)	返回 a 和 b 中的最小值 (支持多种数值类型)	Math.min(10, 20); // 10
取整方法	Math.ceil(x)	向上取整，返回大于等于 x 的最小整数 (double类型)	Math.ceil(3.2); // 4.0Math.ceil(-3.2); // -3.0
	Math.floor(x)	向下取整，返回小于等于 x 的最大整数 (double类型)	Math.floor(3.8); // 3.0Math.floor(-3.8); // -4.0
	Math.round(x)	四舍五入 (float 返回 int, double 返回 long)	Math.round(3.2); // 3Math.round(3.8); // 4
指数与对数	Math.pow(a, b)	返回 a 的 b 次幂	Math.pow(2, 3); // 8.0
	Math.sqrt(x)	返回 x 的平方根 (x ≥ 0)	Math.sqrt(16); // 4.0
	Math.cbrt(x)	返回 x 的立方根	Math.cbrt(8); // 2.0Math.cbrt(-27); // -3.0
	Math.exp(x)	返回 e 的 x 次幂	Math.exp(1); // ≈2.718
	Math.log(x)	返回 x 的自然对数 (以 e 为底, x > 0)	Math.log(Math.E); // 1.0
	Math.log10(x)	返回 x 的以 10 为底的对数 (x > 0)	Math.log10(100); // 2.0
三角函数	Math.sin(rad)Math.cos(rad)Math.tan(rad)	正弦、余弦、正切 (参数为**弧度**)	Math.sin(Math.PI/2); // 1.0Math.cos(Math.PI); // -1.0
	Math.toRadians(deg)	将角度转换为弧度	Math.sin(Math.toRadians(30)); // 0.5 (sin30°)
	Math.toDegrees(rad)	将弧度转换为角度	Math.toDegrees(Math.PI); // 180.0
随机数	Math.random()	返回一个 [0.0, 1.0) 之间的 double 型随机数	(int)(Math.random() * 100) + 1; // 生成[1, 100]的随机整数

所有方法都是静态的，通过 Math.方法名()调用

7.4Math类与Random类

- 7.4.1Math类
- 7.4.2Random类
  - java.util.Random类专门用于生成一个伪随机数

```
import java.io.*;
public class MathExer {
    public static void main(String[] args) {
        int count=0;
        for(int i=0;i<10;i++) {
            int num1,num2,sum=0;
            num1=(int)(Math.random()*10);
            num2=(int)(Math.random()*10);
            System.out.println(num1+"+"+num2+"=?");
            BufferedReader in = new BufferedReader(new
                    InputStreamReader(System.in));
            try {
                sum=Integer.parseInt(in.readLine());
            }catch(Exception e) {
                e.printStackTrace();
            }
            if(num1+num2==sum) {
                System.out.println("回答正确！继续！");
                count++;
            }
            else
                System.out.println("回答错误！");
        }
        System.out.println("共答对了"+count+"道题！");
    }
}
```

```
2+4=?
6
回答正确！继续！
6+3=?
9
回答正确！继续！
5+1=?
6
回答正确！继续！
8+6=?
14
回答正确！继续！
5+6=?
11
回答正确！继续！
0+7=?
7
回答正确！继续！
2+3=?
1
回答错误！
6+6=?
12
回答正确！继续！
1+6=?
7
回答正确！继续！
0+5=?
5
回答正确！继续！
共答对了9道题！
```

7.5日期类

- 7.5.1Date类
- java.util包中的Date类用于表示日期和时间，该类在JDK1.0时就已经开始使用，随着JDK版本的不断升级，Date类中大部分的构造方法都不在使用。

7.5日期类

- 7.5.2Calendar类
- Calendar类是一个抽象类，提供了很多常量，它可以将取得的时间精确到毫秒。

常量字段	含义与说明
Calendar.YEAR	年份
Calendar.MONTH	**月份（注意：0-11 代表 1-12 月）**
Calendar.DATE	一月中的某天（几号），与 Calendar.DAY_OF_MONTH同义
Calendar.DAY_OF_MONTH	一月中的某天（几号）
Calendar.HOUR	小时（12小时制）
Calendar.HOUR_OF_DAY	小时（24小时制）
Calendar.MINUTE	分钟
Calendar.SECOND	秒
Calendar.MILLISECOND	毫秒
Calendar.DAY_OF_WEEK	**一周中的第几天（注意：周日为1，周一为2，以此类推，周六为7）**
Calendar.AM_PM	上午或下午指示器（Calendar.AM或 Calendar.PM）

```java
import java.util.Calendar;
public class MathExer {
    public static void main(String[] args) {
        Calendar c = Calendar.getInstance();
        // 使用get()方法和常量字段获取各部分时间信息
        System.out.println("年份 (YEAR): " + c.get(Calendar.YEAR));
        System.out.println("月份 (MONTH): " + (c.get(Calendar.MONTH) + 1)); // 月份从0开始，需+1
        System.out.println("日 (DATE): " + c.get(Calendar.DATE));
        System.out.println("时 (HOUR_OF_DAY, 24制): " + c.get(Calendar.HOUR_OF_DAY));
        System.out.println("分 (MINUTE): " + c.get(Calendar.MINUTE));
        System.out.println("秒 (SECOND): " + c.get(Calendar.SECOND));
        System.out.println("毫秒 (MILLISECOND): " + c.get(Calendar.MILLISECOND));
        System.out.println("本周第几天 (DAY_OF_WEEK, 周日=1): " + c.get(Calendar.DAY_OF_WEEK));
        System.out.println("本年第几天 (DAY_OF_YEAR): " + c.get(Calendar.DAY_OF_YEAR));
        System.out.println("本年第几周 (WEEK_OF_YEAR): " + c.get(Calendar.WEEK_OF_YEAR));

    }
}
```

```
年份 (YEAR): 2025
月份 (MONTH): 9
日 (DATE): 18
时 (HOUR_OF_DAY, 24制): 9
分 (MINUTE): 48
秒 (SECOND): 27
毫秒 (MILLISECOND): 118
本周第几天 (DAY_OF_WEEK, 周日=1): 5
本年第几天 (DAY_OF_YEAR): 261
本年第几周 (WEEK_OF_YEAR): 38
```

7.5日期类

- 7.5.3DateFormat类
- Java提供了DateFormat类支持日期格式化

方法名	作用描述	返回值
format(Date date)	将 Date对象格式化为指定模式的字符串。	String
parse(String source)	将符合特定格式的字符串解析为 Date对象。**注意：需要处理** ParseException。	Date
getDateInstance()	获取默认格式（只包含日期部分）的 DateFormat 实例。	DateFormat
getDateInstance(int style, Locale locale)	获取指定风格和区域的日期格式器。	DateFormat
getTimeInstance()	获取默认格式（只包含时间部分）的 DateFormat 实例。	DateFormat
getDateTimeInstance()	获取默认格式（只包含时间部分）的 DateFormat 实例。	DateFormat

7.5日期类

- 7.5.3DateFormat类

```
import java.text.DateFormat;
import java.util.Date;
public class MathExer {
    public static void main(String[] args) {
        // 获取当前日期和时间
        Date now = new Date();

        //  SHORT 样式 (日期和时间都使用SHORT)
        DateFormat shortDtf = DateFormat.getDateTimeInstance(DateFormat.SHORT, DateFormat.SHORT);
        System.out.println("SHORT 格式: " + shortDtf.format(now));

        //  MEDIUM 样式 (日期和时间都使用MEDIUM)
        DateFormat mediumDtf = DateFormat.getDateTimeInstance(DateFormat.MEDIUM, DateFormat.MEDIUM);
        System.out.println("MEDIUM 格式: " + mediumDtf.format(now));

        //  LONG 样式 (日期和时间都使用LONG)
        DateFormat longDtf = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG);
        System.out.println("LONG 格式: " + longDtf.format(now));

        //  FULL 样式 (日期和时间都使用FULL)
        DateFormat fullDtf = DateFormat.getDateTimeInstance(DateFormat.FULL, DateFormat.FULL);
        System.out.println("FULL 格式: " + fullDtf.format(now));

        //  混合样式示例：日期使用SHORT，时间使用MEDIUM
        DateFormat mixedDtf = DateFormat.getDateTimeInstance(DateFormat.SHORT, DateFormat.MEDIUM);
        System.out.println("混合(SHORT日期, MEDIUM时间)格式: " + mixedDtf.format(now));
    }
}
```

```
SHORT 格式: 2025/9/18 上午10:02
MEDIUM 格式: 2025年9月18日 上午10:02:05
LONG 格式: 2025年9月18日 CST 上午10:02:05
FULL 格式: 2025年9月18日星期四 中国标准时间 上午10:02:05
混合(SHORT日期, MEDIUM时间)格式: 2025/9/18 上午10:02:05
```

7.5日期类

- 7.5.4SimpleDateFormat类
- 如果想得到特殊的日期显示格式，可以通过DateFormat的子类SimpleDateFormat类来实现。
- 参见P139 例7-20