



主讲教师 张 智  
计算机学院软件工程系  
课程群: 421694618

## 2 Java编程基础

2.1 标识符/关键字/运算符

2.2 数据类型

2.3 基本类型用法

2.4 字符串用法 ★★★

2.5 控制语句

2.6 编程练习

## 2.1 标识符

- Java标识符：以字母、下划线(\_)或美元符号(\$)开头，随后跟随字母、下划线(\_)、美元符号(\$)或数字。
- 标识符是大小写敏感的：
  - 有效的：identifier、userName、User\_name、\_sys\_var1、\$change、\_3\_、OK。
  - 非法的：#name, 25name, &time
  - 标识符不能是关键字和保留字，如：if、class、goto等

# Java关键字

注：51个关键字，2个保留字，都是小写

用于定义数据类型的关键字				
class	interface	enum	byte	short
int	long	float	double	char
boolean	void			
用于定义数据类型值的关键字				
true	false	null		
用于定义流程控制的关键字				
if	else	switch	case	default
while	do	for	break	continue
return				

## Java关键字（续）

用于定义访问权限修饰符的关键字				
private	protected	public		
用于定义类，函数，变量修饰符的关键字				
abstract	final	static	synchronized	
用于定义类与类之间关系的关键字				
extends	implements			
用于定义建立实例及引用实例，判断实例的关键字				
new	this	super	instanceof	
用于异常处理的关键字				
try	catch	finally	throw	throws
用于包的关键字				
package	import			
其他修饰符关键字				
native	strictfp	transient	volatile	assert

还有两个保留字：goto、const

## 新的关键字：var

- Java10引入var用于声明局部变量：

```
var x=100;  
var s="abc";  
for(var i=0; i<10; i++)
```

注意：

- var变量必须在定义时初始化： var x; x=100; ✗
- var变量初始化后就确定数据类型，之后不能修改： var s="abc"; s=100; ✗
- 类成员变量、方法的参数、返回类型也不能使用var

## Java运算符

注意：Java没有sizeof运算，也没有&amp;取地址运算

优先级	符号	名称	结合性(与操作数)	目数	说明
1	.	点	从左到右	双目	
	( )	圆括号	从左到右		
	[ ]	方括号	从左到右		
2	+	正号	从右到左	单目	
	-	负号	从右到左	单目	
	++	自增	从右到左	单目	前缀增，后缀增
	--	自减	从右到左		前缀减，后缀减
	~	按位非/取补运算	从右到左	单目	
	!	逻辑非	从右到左	单目	
3	*	乘	从左到右	双目	
	/	除	从左到右	双目	整数除法，取商的整数部分，小数部分去掉。 不四舍五入
	%	取余	从左到右	双目	浮点数也可以取余
4	+	加	从左到右	双目	
	-	减	从左到右	双目	
5	<<	左移位运算符	从左到右	双目	
	>>	带符号右移位运算符	从左到右	双目	若符号位为正，则在最高位插入 0；若符号位为负，则在最高位插入 1
	>>>	无符号右移	从左到右	双目	无论正负，都在最高位插入 0
6	<	小于	从左到右	双目	
	<=	小于或等于	从左到右	双目	
	>	大于	从左到右	双目	
	>=	大于或等于	从左到右	双目	
	instanceof	确定某对象是否属于指定的类	从左到右	双目	

优先级	符号	名称	结合性(与操作数)	目数	说明
7	==	等于	从左到右	双目	
	!=	不等于	从左到右	双目	
8	&	按位与	从左到右	双目	无论左边是 true 还是 false, 右边都运算
9		按位或	从左到右	双目	两边都参与运算
10	^	按位异或	从左到右	双目	同为 false, 异为 true
11	&&	短路与	从左到右	双目	当左边为 false 右边不运算, 结果一定为 false
12		短路或	从左到右	双目	当左边为 true, 右边不运算结果一定为 true
13	? :	条件运算符	从右到左	三目	
14	=	赋值运算符	从右到左	双目	
	+=	混合赋值运算符			
	-=				
	*=				
	/=				
	%=				
	&=				
	=				
	^=				
	<<=				
	>>=				

【返回】

## 2.2 数据类型

注：Java没有unsigned关键字

类别	类型	关键字	包装类	字节数	取值范围
基本数据类型	整数	byte	Byte	1	$-2^7 \sim 2^7-1$ (-128..127)
		short	Short	2	$-2^{15} \sim 2^{15}-1$ (-32768 ~ 32767)
		int	Integer	4	$-2^{31} \sim 2^{31}-1$
		long	Long	8	$-2^{63} \sim 2^{63}-1$
	浮点数	float	Float	4	略
		double	Double	8	
	字符型	char	Character	2	$0 \sim 2^{16}-1$ (0 ~ 65535)
引用类型	布尔型	boolean	Boolean	未确定	true、false
	数组	[]			
	类	class			
	接口	interface			



## 包装类概念

- Java语言不把基本数据类型的变量视为对象
- 为了满足万物皆对象的理念，需要对基本数据类型的变量进行封装处理变成对象，而负责这项任务的相关类，称为**包装类** (wrapper class)

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

包装类封装在java.lang包中

## 基本类型和包装类区别

- 定义不同：包装类是对象，基本数据类型不是
- 使用方式不同：基本数据类型可以直接赋值，包装类一般需要用new来分配存储空间，在泛型集合中必须使用包装类，不能使用基本数据类型，如：List<Integer>
- 初始值不同：基本类型的初始值如int为0，boolean为false，而包装类型的初始值为null
- 存储方式和位置不同：基本数据类型存储在栈中(stack)，包装类型是将对象放在堆中(heap)，然后通过引用来使用(引用放在stack中)

## 包装类特点

- 基本类型不能赋值为null，如： `int x = null;` ✕
- 但包装类型可以，如： `Integer x = null;` ✓
- 包装类有自己的属性(公有静态常量)和方法，而基本数据类型则没有：
  - `Integer.MAX_VALUE`：返回int最大值(2147483647)
  - `Integer.SIZE`：返回int类型位数(32，1字节=8位) ← 注：Boolean类没有SIZE属性
  - `Integer.parseInt("数字串")`：将数字串转为int值
  - `Integer.toString(int)`：将int转为字符串

## 补充：包装类常用常量(属性)和方法 -- 以Integer类为例

常量类型和名称	功能介绍
<code>public static final int MAX_VALUE</code>	表示int类型可以描述的最大值，即 $2^{31}-1$
<code>public static final int MIN_VALUE</code>	表示int类型可以描述的最小值，即 $-2^{31}$
<code>public static final int SIZE</code>	表示int类型采用二进制补码形式的位数
<code>public static final int BYTES</code>	表示int类型所占的字节个数
<code>public static final Class TYPE</code>	表示int类型的Class实例

方法声明	功能介绍
<code>Integer(int value)</code>	根据参数指定的整数来构造对象（已过时）
<code>Integer(String s)</code>	根据参数指定的字符串来构造对象（已过时）
<code>int intValue()</code>	获取调用对象中的整数值并返回
<code>static Integer valueOf(int i)</code>	根据参数指定整数值得到Integer类型对象
<code>boolean equals(Object obj)</code>	比较调用对象与参数指定的对象是否相等
<code>String toString()</code>	返回描述调用对象数值的字符串形式
<code>static int parseInt(String s)</code>	将字符串类型转换为int类型并返回
<code>static String toString(int i)</code>	获取参数指定整数的十进制字符串形式
<code>static String toBinaryString(int i)</code>	获取参数指定整数的二进制字符串形式
<code>static String toHexString(int i)</code>	获取参数指定整数的十六进制字符串形式
<code>static String toOctalString(int i)</code>	获取参数指定整数的八进制字符串形式

## 装箱和拆箱

老的写法: `Integer n = new Integer(10);`

`int a;` //基本类型

`Integer n = Integer.valueOf(10);` //Integer对象

`a = n;` // ok, 拆箱操作 `a=10` → 拆箱操作: 包装类对象→值类型

`Integer m = 10;` // ok, 装箱操作 → 装箱操作: 值类型→包装类对象

`int b = m.intValue();` // `b=10`, `intValue`返回Integer对象的整数值

`m.equals(n);` // true 值比较

注: 装箱、拆箱会损耗性能

注意: 使用`equals`比较值, 不要使用`==`

## 关于：无符号整型处理

注：Java没有unsigned关键字，但有相应的unsigned函数

- 例如：byte是有符号整型，范围是：-128~+127
- 如果想把byte看作无符号整型，它的范围应该就：0~255
- 如何把一个byte按无符号整数转换为int？



```
byte a = -1;  
int x = Byte.toUnsignedInt(a);    // x= 255  
  
byte b = -128;  
int y = Byte.toUnsignedInt(b);    // y= 128
```

将负数的补码全部看成有效数字

## 关于：大数据 BigInteger 和 BigDecimal

- **BigInteger**：任意大小的整数（内部使用一个int数组来存储大整数）
- **BigDecimal**：任意大小且精度完全准确的浮点数

```
BigInteger a = new BigInteger("12345678901234567890");
```

 ← 注：大数据要用字符串处理的

```
BigInteger b = new BigInteger("1234567890");
```

```
BigInteger sum = a.add(b); // 加减乘除分别为：add、subtract、multiply、divide
```

- **BigInteger**的转换：

```
Long x = 1234567890L;
```

```
BigInteger y = BigInteger.valueOf(x);
```

```
Long z = y.longValue(); // 注：数据如果超出long范围会溢出
```

```
import java.math.BigInteger;
```

[【返回】](#)

## 2.3 基本类型用法

- [变量和常量](#)
- [数据输出格式控制](#)
- [整数/浮点型](#)
- [boolean类型](#)
- [char类型](#)

[【返回】](#)



## 1. 变量和常量

(1) 方法中的变量必须经初始化后才能被使用：

...

```
int x = (int)(Math.random() * 100);    // Math.random(): 产生[0,1)之间随机数
```

```
int y, z;    // y, z 只是声明, 尚未初始化
```

```
if (x > 50)
```


```
{
```

```
    y = 1;
```

```
}
```

```
z = y + x;
```

...



编译会报错: Variable 'y' might not have been initialized

## (2) 类成员变量初始化:

- Java允许在类定义时直接初始化成员变量
- 但一般都是通过构造函数进行成员变量初始化
- 如果以上都没有, 则在对象创建时被自动初始化:
  - 数值型成员: 默认值为0, 如int为0, double为0.0
  - boolean成员: 默认值为false值
  - char成员: 默认值为'\u0000' (其实也是0, 空字符)
  - 所有对象变量(如String): 默认值为null

## 类成员变量初始化示例

```
public class Main {  
    private int i = 100; ←----- 成员变量可直接初始化
```

```
    private int j;           //默认值为0
```

```
    private boolean f;      //默认值为false
```

```
    public Main() { ←----- 默认构造函数  
    }  
}
```

```
    public Main(int i, int j, boolean f) { ←----- 全参构造函数  
        this.i = i;  
        this.j = j;  
        this.f = f;  
    }  
}
```

```
    public void display() {  
        System.out.println( "i=" + i );  
        System.out.println( "j=" + j );  
        System.out.println( "f=" + f );  
    }  
}
```

```
    public static void main (String args[]) {  
        Main t1 = new Main();  
        t1.display();  
        Main t2 = new Main(1,2,true);  
        t2.display();  
    }  
}
```

运行结果

```
i=100  
j=0  
f=false  
i=1  
j=2  
f=true
```

## 常量（符号常量）

使用final来定义符号常量(一般用大写字符):

- 定义符号常量: `final double PI = 3.14159;` ← 变量声明前面加final变常量
- 常量可先声明后赋值: `final double PI; PI = 3.141519;` ✓
- 常量赋值后就不可再修改: `PI = 3.14;` ✗ ← final常量不可修改
- 系统提供的符号常量: 一般是公有的静态常量 `public static final`, 可以直接通过类名调用  
如: `Integer.MIN_VALUE`、`Double.MAX_VALUE`、`Character.SIZE`

[【返回】](#)

## 2. 数据输出格式的控制

- 方法1: `String.format("格式串", 数据)`
- 方法2: `System.out.printf("格式串", 数据)`

← 推荐使用

```
int a = 100;  
double b = 123.456;  
System.out.println( String.format("a=%-5d,b=%.2f", a,b) );  
System.out.printf( "a=%05d,b=%10.2f \n", a,b );
```

运行结果

```
a=100 ,b=123.46  
a=00100,b=      123.46
```

[【返回】](#)

### 3. 整数/浮点型

#### ■ 整数类型：byte、short、int、long

如：17 (+), 0b10 (二), 017 (八), 0xff00 (十六), 17L (长整型)

#### ■ 浮点数：float、double

如：3.14、3.14f、3.02e23、123.4D (类型字符大小写均可)

注意：

float a = 3.4f;    ✓ 'f' 不能省略

float a = 3.4;    ✗

double a = 3.4;    ✓

说明：Java中普通的浮点数默认是double类型

## 算术运算注意

- 自增与自减运算符只适用于变量，如：10++; ✕
- 两个整数类型的数据做除法时，结果只保留整数部分： $3/2 = 1$ ， $-9/2 = -4$
- %取余运算不仅能用于整型，也能用于浮点类型：

$$9\%2 = 1 \quad 9.5\%2 = 1.5 \quad -9\%2 = -1$$

余数可为浮点数

余数可为负数

↑  $3.0/2 = 1.5$

## 算术运算注意

- 整数除0时会报错
- 浮点数除0时，不会报错，会返回几个特殊值：

```
int i = 1/0;           // 报错  
  
double d1 = 0.0 / 0;   // NaN    表示Not a Number (Double.NaN)  
  
double d2 = 1.0 / 0;   // Infinity 表示无穷大 (Double.POSITIVE_INFINITY)  
  
double d3 = -1.0 / 0;  // -Infinity 表示负无穷大 (Double.NEGATIVE_INFINITY)
```



## 浮点数相等的判断

- 浮点数在计算机中常常无法精确表示，并且计算可能出现误差，因此，判断浮点数相等不要用==判断
- 正确方法：利用差值绝对值小于某个临界值来判断

```
double x = 1.0 / 10;           // x = 0.1
double y = 1 - 9.0 / 10;       // y = 0.09999999999999998
System.out.println( x==y );    // false
// 所以用 y==0.1 不靠谱，正确用法：
if ( Math.abs(y - 0.1) < 0.00001 ) { // 采用临界值法判断相等
    // y 可近似认为是 0.1
}
```

## 整数运算溢出问题

- 整数由于存在范围限制，如果计算超出了范围，就会产生溢出
- 溢出不是报错，只是结果不符合预期

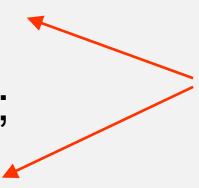
```
short x = 32767;
```

```
x++; // -32768
```

```
short y = -32768;
```

```
x--; // 32767
```

思考一下why?



## 整数移位运算

注：对byte和short会首先转换为int再进行位移

- `>>`：右移，正数高位补0，负数高位补1（带符号右移）
- `<<`：左移，并且在低位补0
- `>>>`：无符号右移，无论正负高位都补0

```
int n=7;           // n=7    00000000 00000000 00000000 00000111
int a=n<<2;        // a=28    00000000 00000000 00000000 00011100
int b=n>>2;        // b=1     00000000 00000000 00000000 00000001
int m=-7;          // m=-7    11111111 11111111 11111111 11111001  负数补码
int c=m<<2;        // c=-28   11111111 11111111 11111111 11100100  负数补码
int d=m>>2;        // d=-2    11111111 11111111 11111111 11111110  负数补码
m>>>2;            //无符号右移 00111111 11111111 11111111 11111110
```

一般情况：左移`<<n`位：相当于 乘以  $2^n$ ，右移`>>n`位：相当于 整除  $2^n$

## 整数位运算

- 位运算是按位进行与、或、非和异或的运算
- 两个整数位运算：按位对齐，然后依次对每位进行位运算

```
int n=7;      // n=7    00000000 00000000 00000000 00000111
int m=-7;     // m=-7    11111111 11111111 11111111 11111001  负数补码

与: n&m      // 1       00000000 00000000 00000000 00000001
或: n|m      // -1      11111111 11111111 11111111 11111111
异或: n^m    // -2      11111111 11111111 11111111 11111110
非: ~m       // 6       00000000 00000000 00000000 00000110
```

## 类型自动提升和强转

- 参与运算的两个数类型不一致，那么计算结果以较大类型为准
- 浮点数强转为整数时小数部分丢弃
- 尽量避免将大范围的整数转型为小范围的整数，以免数据丢失或错误

```
short s = 1234;
int i = 123456;
int x = s + i;    // 124690  s自动转型为int
short y = s + i;  // 编译错误!
int p = 32768;    // 32768: 00000000 00000000 10000000 00000000
short q = (short)x; // -32768: 10000000 00000000 负数补码
int n = (int)-12.3; // -12
int m = (int) (12.7 + 0.5); // 13
```

[【返回】](#)

## 4. boolean类型

- boolean有两个值：true 和 false（小写）
- **特别注意：** 数字值不能自动转换为boolean

```
int x = 1;

if ( x ) { }           // 报错：Type mismatch:cannot convert from int to boolean

if ( x !=0 ) { }       // OK，显示写出条件

boolean flag = true;

if( flag ) { }         //OK，条件使用布尔量
```

**短路运算：** 如果一个布尔运算表达式能提前确定结果，则后续的计算不再执行，直接返回结果

例如：false&& x, true|| x : x被短路不参与运算

[【返回】](#)

## 5. char类型

- 一个char代表一个16bit无符号的Unicode字符

Unicode称为统一码或万国码，它为每种语言的每个字符设定了唯一二进制编码以满足跨语言、跨平台的文本处理要求，utf-8、utf-16都是Unicode规范的实现

- 一般用法：

- 正确：char ch1='a', ch2='\n', ch3='中';

注：汉字也是一个Unicode字符

- 错误：char ch1='abc', ch2="a";

注：char类型使用单引号，且仅有一个字符

- 标准写法：

char ch='\u0041'; 等价于 ch = 'A';

以 \u 开头 + 4个16进制数字

注：必须是4位，'\u41' ×

## 常用的转义符

转义符	描述
\n	换行
\r	回车
\t	制表符
\b	退格
\0	空字符(NULL)
\'	单引号
\"	双引号
\\	反斜杠
\ddd	1到3位八进制数所代表的字符
\xhh	1到2位十六进制所代表的字符



## char 和 int 之间的转换

■ char和int可以转换: char范围 [0-65535]

```
char ch1 = 65;
```

```
int ch2 = 'A';
```

```
System.out.println(ch1);      // 输出 A
```

```
System.out.println(ch2);      // 输出 65
```

```
System.out.println((int)ch1); // 输出 65
```

```
System.out.println((char)ch2); // 输出 A
```

## 字符的检测方法：返回值为 true 或 false

- `Character.isLetter(char)` -- 指定字符是否是字母
- `Character.isDigit(char)` -- 指定字符是否是数字
- `Character.isLetterOrDigit(char)` -- 指定字符是否是字母或数字
- `Character.isWhitespace(char)` -- 指定字符是否是空格
- `Character.isLowerCase(char)` -- 指定字符是否是小写字母
- `Character.isUpperCase(char)` -- 指定字符是否是大写字母

char类型的包装类

- `Character.toUpperCase(char)`: 字母小写变大写
- `Character.toLowerCase(char)`: 字母大写变小写

[【返回】](#)

## 2.4 字符串用法 -- String类 ★★★

- String不是基本类型，而是一个class类
- 用双引号 "..." 表示字符串
- 与C和C++不同，String不用 '\0' 作为结束

创建方法：

方法1: `String s1 = "hello";` ←----- 注: "Hello"是一个字符串常量

方法2: `String s1 = new String("hello");` //使用构造函数创建

`String s2 = "";` ←----- 空串，长度为0的串

`String s3 = null;` ←----- 空对象(没有分配内存空间)

## 注意：String两种创建方法的差异

```
String s1 = "Hello";
```

```
String s2 = "Hello";
```

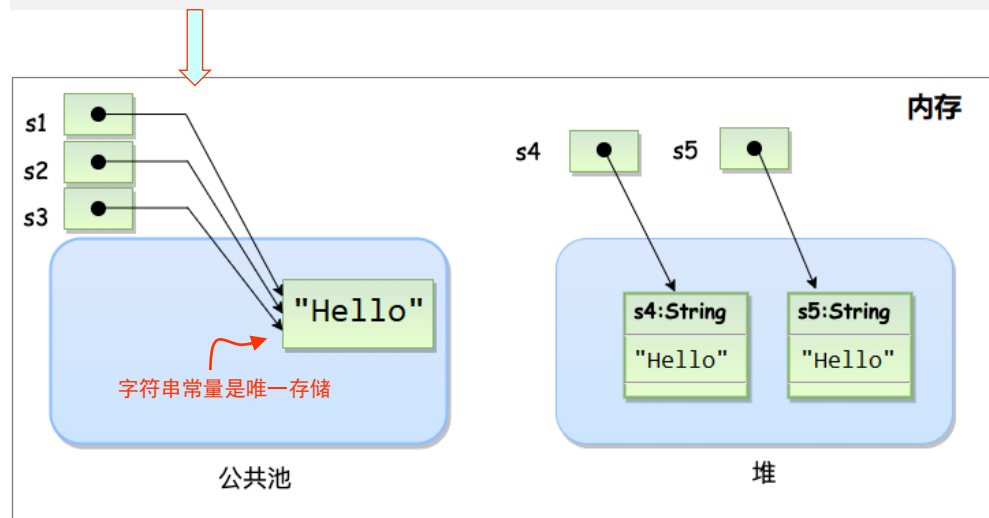
注："Hello"是一个字符串常量，意味着在内存中只有一个，即 s1,s2两个对象引用的地址是一样的

```
String s3 = s1; // s1, s2, s3都是相同引用（如下图所示）
```

```
String s4 = new String("Hello");
```

```
String s5 = new String("Hello");
```

注：使用new创建的字符串，无论值是否相同，都会分配不同内存空间，即s4,s5引用的地址不相同



## 字符串的常用操作

1. 字符比较
2. 字符串长度和字符获取
3. 字符串连接
4. 字符串分割
5. 字符串查找
6. 获取子串
7. 字符串判断
8. 字符串匹配
9. 字符串替换
10. 字符串去空白
11. 字符串转换
12. String、StringBuffer、StringBuilder

## 1. 字符串比较:

忽略大小写的比较: equalsIgnoreCase()

- `equals()` : 比较字符串的内容是否相同
- `==` : 比较的是不是同一个引用(地址)

```
String s1 = "abc";  
String s2 = "abc";  
System.out.println( s1==s2 );           // true  
System.out.println( s1.equals(s2) );     // true
```

强调: 字符串内容比较是否相同时要用equals() 方法, 不要用 ==

```
String s3 = new String("abc");  
String s4 = new String("abc");  
System.out.println( s3==s4 );           // false  
System.out.println( s3.equals(s4) );     // true
```

## 字符串比较：

### ■ `compareTo(String s)`：字典比较

- 如果原串小于s串则返回负值，如果原串大于s串则返回正值，如果原串等于s串则返回零
- 如果比较字符串有包含关系，返回的值是它们长度的差值
- **一般比较过程**：先比较第一个元素，如果相等再比较第二个元素...，返回**元素之间的差值**

```
String s1 = "abc";  
String s2 = "ABCD";  
  
int x = s1.compareTo(s2);           // x=32, 'a' - 'A' = 32  
  
int y = "abcdef".compareTo(s1)      // y=3  包含关系，返回长度差
```

## 2. 字符串长度和字符获取

■ `length()`: 获取串的长度（字符个数）

■ `charAt(int index)`: 获取指定索引位置处的字符(char) ← index索引值从0开始

```
String s="student";
```

```
s.length()      //7
```

```
s.charAt(2)     // 'u'
```

字符串不能用下标形式: `s[2]` ✕

```
"中国".length() //2
```

```
"ABC\\n\\u4e2d\\u6587".length() //12 想一想
```

```
String s="time over";
```

```
for( int i = s.length() - 1; i >=0; i-- ) {
```

```
    System.out.print( s.charAt(i) );
```

逆序输出串

```
}
```



### 3. 字符串连接:

```
String s1 = "Hello";
```

```
String s2 = " Java";
```

```
String s3 = s1 + s2;           // s3 = "Hello Java"
```

```
String s4 = s1.concat(s2);     // s4 = "Hello Java" → 注: s1串自身并没有改变
```

```
int i = 10;
```

```
String s5 = "i=" + i;         // s5 = "i=10"
```

说明: 字符串与其他类型进行“+”运算时, 系统将自动将其转换为字符串进行连接

```
String[] arr = { "A", "B", "C" }; //字符串数组
```

```
String s6 = String.join("-", arr); // s6 = "A-B-C"
```

↑  
用指定串连接字符串数组

## 程序阅读：

```
int a = 1, b=2;  
  
String str = "hello";  
  
System.out.println( str+a+b );    // hello12  
  
System.out.println( a+b+str );    // 3hello
```

- **思考题：**将下面 int 值视为字符，如何将它们连接拼成一个字符串，结果又是多少？

```
int a = 65;   int b = 97;   int c = 48;
```

- **答案：**String s = "" + (char)a + (char)b + (char)c; 结果 s = "Aa0"

## 4. 字符串分割

注：分割不会改变原字符串的内容，而是返回了一个新字符串

■ `split(String s)`：根据给定s拆分字符串（返回字符串数组）

```
String s1="12:25:30";
```

```
String[ ] arr1 = s1.split(":"); // arr1 = {"12", "25", "30"}
```

```
String s2 = "wust.edu.cn";
```

```
String[ ] arr2 = s2.split("\\."); // arr2 = {"wust", "edu", "cn"}
```

注：点号比较特殊，要用"\\."表示

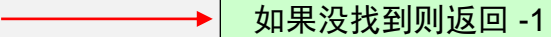
## 5. 字符串查找

- `indexOf(String s)`: 获取指定字符串第一次出现的索引值
- `indexOf(String s, int fromIndex)`: 从指定索引位置开始, 获取指定字符串第一次出现的索引值
- `lastIndexOf(String s)`: 获取指定字符串最后一次出现的索引值

```
String s="I am a student";
```

```
s.indexOf("stu")    // 7 索引值从0开始
```

```
s.indexOf("you")    // -1 没找到
```



```
s.indexOf("a", 3)    // 5 从第4个开始找
```

```
s.lastIndexOf("a")  // 5 找最后一个"a"
```

## 6. 获取子串

- `substring(int start)`: 从指定位置开始一直截取到末尾
- `substring(int start, int end)`: 截取 `[start,end-1]` 范围子串

```
String s1="time is over";  
String s2=s.substring(2)      // s2="me is over"  
String s3=s.substring(1, 3)   // s3="im"
```

## 7. 字符串判断

- `contains(String s)`: 是否包含指定字符串
- `startsWith(String s)`: 是否以指定的字符串开头
- `endsWith(String s)`: 是否以指定的字符串结尾
- `isEmpty()`: 判断是否为空串

```
String s="I am a student";  
s.contains("stu")    // true  
s.startsWith("ent") // false  
s.endsWith("ent")   // true  
s.isEmpty()         // false  
"".isEmpty();       // true, ""是长度为0的空串(注:不是null)  
" ".isEmpty();      // false, " "是长度不为0的空白串
```

## 8. 字符串匹配

正则式：是一种"规则字符串"，用于匹配字符串中字符组合的模式

■ `matches(String reg)`：检查字符串是否与给定的正则式匹配

```
String reg = "[0-9]+$"; // 定义一个正则式(字符串)
```

```
"789".matches(regex) // true
```

```
"00123".matches(regex) // true
```

```
"123a".matches(regex) // false
```

```
"98 41".matches(regex) // false
```

正则式 "[0-9]+\$" 含义：

- ^表示开头，\$表示结束
- [0-9] 表示：匹配所有数字字符 (0~9)
- +表示：1次或多次
- "[0-9]+\$"表示：匹配数字，1个或多个，即数字串(包括0开头的)

## 9. 字符串替换

注：替换不会改变原字符串的内容，而是返回了一个新字符串

■ `replace(String oldS, String newS)`：用新串newS替换老串oldS

```
String s1 = "time over";  
String s2 = s1.replace("e", "yy"); // s2="timyy ovyyr", s1不变
```



## 10. 字符串去空白

注：不会改变原字符串的内容，而是返回了一个新字符串

■ `trim()`：去掉前后空白字符，包括：空格，`\t`，`\r`，`\n`

```
String s1 = " \tHello\r\n ";
```

```
String s2 = s1.trim();           // s2 = "Hello"   s1保持不变
```

## 11. 字符串转换

注：转换也不会改变原字符串的内容，而是返回了一个新字符串

- `getBytes()`：将字符串转成字节数组 (`byte[]`)，默认Unicode编码值)
- `toLowerCase()`：大写字母变成小写
- `toUpperCase()`：小写字母变成大写

```
String s="Hello";  
  
byte[] s1 = s.getBytes();           // [72, 101, 108, 108, 111] 'H'→72 ...  
  
String s2 = s.toUpperCase();         // "HELLO"  
  
String s3 = s.toLowerCase();         // "hello"
```

注：s1串本身都没有改变

## 字符串转换（续）

- `toCharArray()`: 将字符串转成字符数组
- `valueOf(Object obj)`: 将任意类型转成字符串

用法见下页汇总

## 转换汇总：

### ■ 字符串 → 数值：以int/Integer为例，其他类型类似

#### (1) String转换为int：

```
int i = Integer.parseInt("123");    // i=123，默认为十进制
```

```
int n = Integer.parseInt("ff", 16); // n=255，按十六进制转换
```

#### (2) String转换为Integer：

```
Integer x = Integer.valueOf("123"); // 串转换为Integer对象
```

注意：转换时对数据格式要求严格

```
int i = Integer.parseInt("123.4");  ×
```

```
double d = Double.parseDouble("1.23a"); ×
```

均会报错：NumberFormatException

## ■ 数值 → 字符串：

方法1：使用String.valueOf()方法

```
String s = String.valueOf(123);    // value可为任意类型，包括Object
```

方法2：使用toString()方法

```
String s = Integer.toString(123);  // 其他类型类似
```

方法3：使用"+"连接 (最直接)

```
String s = "" + value;             // value为任意数值类型
```

## 转换汇总：

### ■ 字符数组 → 字符串

使用String构造函数转换

```
char[ ] charArr = { 'a', 'b', 'c' };    // 字符数组  
String str = new String(charArr);    // str = "abc"
```

### ■ 字符串 → 字符数组

使用toCharArray()方法转换

```
String str = "abc";  
char[ ] c= str.toCharArray();    // c=[ 'a', 'b', 'c' ]
```

补充：Arrays.toString(一维数组)：可以将任意类型一维数组变成字符串

例如：System.out.print( Arrays.toString(charArr) ); //输出结果 [a, b, c]

## 12. 关于 String、StringBuffer、StringBuilder

- String字符串是不可改变的，也就是说一旦创建了String对象，那它的值就无法改变！
- 要对字符串自身进行修改，可使用StringBuffer类
- StringBuffer对象能被多次修改，且不产生新的对象！
- StringBuilder和StringBuffer本质上没区别，只是StringBuilder去掉了线程安全，不同步，效率更高。

## StringBuffer常用方法

这些操作都将修改StringBuffer对象自身

方法	功能
append(String s)	将s追加到尾部
reverse()	字符串内容反转
int capacity():	获取容量（不是length值）
setCharAt(int index, char ch):	在指定位置替换一个字符
deleteCharAt(int index):	在指定位置删除一个字符
delete(int start, int end):	删除start到end-1位置的子串
insert(int offset, int i):	将整数 i 的字符串形式插入到offset位置
replace(int start, int end, String str):	将start至end-1位置内容替换为str



## StringBuffer用法示例

```
StringBuffer sb = new StringBuffer("Hello");    // StringBuffer sb = "Hello"; ✕  
sb.append(" world").append("!");                // sb被修改, sb = "Hello world!"
```

## String与StringBuilder的相互转换

### ■ String → StringBuilder:

```
String s = "hello";
```

```
StringBuilder sb = new StringBuilder(s);
```

### ■ StringBuilder → String:

```
StringBuilder sb = new StringBuilder();
```

```
sb.append("abc").append("efg");
```

```
String s = sb.toString();    // s="abcefg"
```

## StringBuffer用法示例

```
public int reverse(int n) {  
    StringBuffer sb = new StringBuffer( String.valueOf(n) ).reverse();  
    return Integer.parseInt( sb.toString() );  
}
```

逆序整数n，不使用分解

## StringBuffer串的比较

- StringBuffer串的比较，不能直接使用equals()
- 应该先使用toString()方法将StringBuffer对象转换为String字符串，再使用equals()方法比较：

```
sb1.toString().equals( sb2.toString() )
```

## StringBuffer用法示例

```
public boolean isPalindrome(String s) {  
    StringBuffer sb = new StringBuffer(s);  
    return sb.reverse().toString().equals(s);  
}
```

判断s是否是回文串

[【返回】](#)

## 2.5 控制语句

- 条件语句
  - ✓ if
  - ✓ if...else...
  - ✓ 布尔表达式 ? 语句1:语句2
  - ✓ switch
- 循环控制语句
  - ✓ for
  - ✓ while
  - ✓ do...while
- 中断语句:
  - ✓ break、continue、return

## switch语句

可以是 int、byte、short、char 或 String、枚举 类型，不允许使用浮点或long表达式

```
switch (expression) {
```

```
    case 值1: ...
```

```
        break;
```

```
    case 值2: ...
```

```
        break;
```

```
    default: ...
```

```
        break;
```

```
}
```

如没break，则程序的执行将继续到下一个case(穿透性)

不允许有  
重复case值

## switch示例1：

```
public class Main {  
    public static String generate(String name, String gender) {  
        String title = "您好！";  
        switch (gender) {  
            case "男":  
                title += name + "先生";  
                break;  
            case "女":  
                title += name + "女士";  
                break;  
            default:  
                title += name;  
        }  
        return title;  
    }  
    public static void main(String[] args) {  
        System.out.print("请姓名=");  
        String name = new Scanner(System.in).nextLine();  
        System.out.print("请性别=");  
        String gender = new Scanner(System.in).nextLine();  
        System.out.print(generate(name, gender));  
    }  
}
```

运行结果

请姓名= 小明  
请性别= 男  
您好！ 小明先生

请姓名= 小明  
请性别=  
您好！ 小明



## switch示例2:

```
public class Main {  
    public static void main(String[] args) {  
        while (true) {  
            System.out.println("请输入成绩(输入-1结束): ");  
            int score = new Scanner(System.in).nextInt();  
            if (score == -1) {  
                System.out.println("再见");  
                break;  
            }  
            int a = score / 10;  
            switch (a) {  
                case 10:  
                case 9:  
                    System.out.println("优秀");  
                    break;  
                case 8:  
                    System.out.println("良好");  
                    break;  
                case 7:  
                    System.out.println("中等");  
                    break;  
                case 6:  
                    System.out.println("及格");  
                    break;  
                default:  
                    System.out.println("不及格");  
                    break;  
            } // end switch  
        } // end while  
    } // end main  
} // end class
```

### 运行结果

```
请输入成绩(输入-1结束):  
90  
优秀  
请输入成绩(输入-1结束):  
60  
及格  
请输入成绩(输入-1结束):  
-1  
再见
```

## while/do...while/for

```
int x=1;
while( x<10 ) {
    ...
    x++;
}
```

```
int x;
for( x=1; x<10; x++ ) {
    ...
}
```

```
int x=1;
do {
    ...
    x++;
} while( x<10 ) ;
```

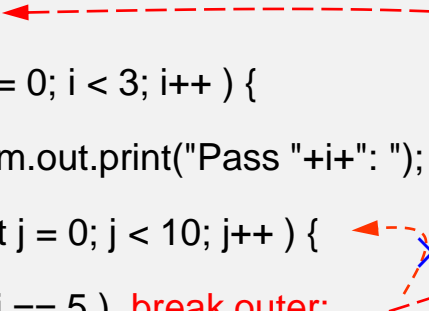

```
for( int x=1; x<10; x++ ) {
    ...
}
```

此时x的作用域只在循环内

## break、continue

- **特殊用法：** break、continue后接标签名
- **什么是标签：**
  - 标签可标识控制需要转换到的任何有效语句(块)
  - 标签用法：  
**标签: 语句(块)**      即在有效语句块前加个"标签名:"

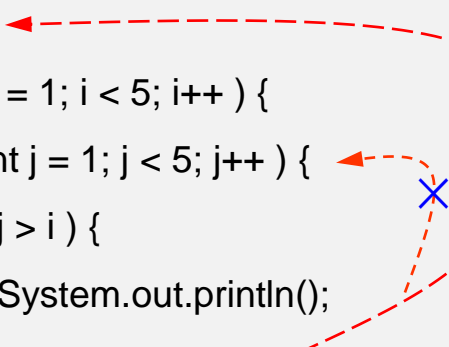

## 标签示例1

```
public class Main {  
    public static void main(String args[]){  
        outer:   
        for( int i = 0; i < 3; i++ ) {  
            System.out.print("Pass "+i+": ");  
            for( int j = 0; j < 10; j++ ) {   
                if ( j == 5 ) break outer;  
                System.out.print(j+" ");  
            }  
            System.out.println();  
        }  
    }  
}
```

运行结果

Pass 0: 0 1 2 3 4

## 标签示例2

```
public class Main {  
    public static void main(String args[]){  
        outer:   
        for( int i = 1; i < 5; i++ ) {  
            for( int j = 1; j < 5; j++ ) {   
                if( j > i ) {  
                    System.out.println();  
                    continue outer;  
                }  
                System.out.print(" " + ( i * j ) );  
            }  
            System.out.println();  
        }  
    }  
}
```

运行结果

1
2 4
3 6 9
4 8 12 16

[【返回】](#)

## 2.6 编程练习

1. 字符统计: 统计字符串中大写字母、小写字母、数字、空格以及其他字符个数
2. 回文判断: 如"abba"、"abcba"是回文, "abcd"不是回文
3. 随机整数: 生成一个区间在 [MIN, MAX] 的随机整数
4. 素数: 输出1-100之间的所有素数, 并统计个数
5. 水仙花数: 一个三位数, 其各位数字立方和等于该数
6. 计算圆周率 $\pi$ :  $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots (-1)^{m+1} \frac{1}{2m-1} + \cdots$
7. 递归: 求  $2+4+6+\dots+100$ 之和
8. 最小值和最大值: 使用Math类

[【返回】](#)

## 1. 字符串统计

Ex\_1.java

```
public class Ex_1 {  
    //定义计数器  
    private static int lower = 0;  
    private static int upper = 0;  
    private static int number = 0;  
    private static int space = 0;  
    private static int other = 0;  
  
    //统计  
    public static void count(String s) {  
        for (int i = 0; i < s.length(); i++) { ◀--- 遍历字符串  
            char ch = s.charAt(i);  
            if (Character.isLowerCase(ch))  
                lower++;  
            else if (Character.isUpperCase(ch))  
                upper++;  
            else if (Character.isDigit(ch))  
                number++;  
            else if (Character.isWhitespace(ch))  
                space++;  
            else  
                other++;  
        }  
    }  
  
    //主程序  
    public static void main(String[] args) {  
        System.out.print("请输入字符串:");  
        String s = new Scanner(System.in).nextLine(); ◀--- 读取一行字符串(以回车结束)  
        count(s);  
        System.out.println("小写字母个数:" + lower +  
            ",大写字母个数:" + upper +  
            ",数字个数:" + number +  
            ",空格个数:" + space +  
            ",其他字符个数:" + other);  
    }  
}
```

运行结果

请输入字符串:ab12# 8\$YZ go!

小写字母个数:4,大写字母个数:2,数字个数:3,空格个数:2,其他字符个数:3

【返回】

## 2. 回文判断

```
public class Ex_2 {  
    //判断回文(普通实现)  
    public static boolean isPalindrome(String s) {  
        char ch1, ch2;  
        for( int i = 0; i < s.length()/2; i++ ) {  
            ch1 = s.charAt(i);  
            ch2 = s.charAt( s.length()-1-i );  
            if ( ch1 != ch2 ) {  
                return false;  
            }  
        }  
        return true;  
    }  
}
```

一前一后比对

//判断回文(StringBuffer实现)

```
public static boolean isPalindrome(String s) {  
    StringBuffer sb = new StringBuffer(s);  
    return sb.reverse().toString().equals(s);  
}
```

//主程序

```
public static void main(String[] args) {  
    System.out.print("请输入字符串:");  
    String s = new Scanner(System.in).nextLine();  
    System.out.println(isPalindrome(s));  
}
```

Ex\_2.java

运行结果

请输入字符串:abcCcba  
true

请输入字符串:abcd  
false

[【返回】](#)



### 3. 随机整数：区间在 [min, max]

Ex\_3.java

```
public class Ex_3 {
```

```
    //方法1：使用Math.random()
```

```
    public static int rnd( int min, int max ) {
```

```
        double x = Math.random(); // [0,1)
```

```
        double y = x * (max - min + 1) + min; // 变换到 [min,max+1)
```

```
        int n = (int) y; // 强转为整数[min,max]
```

```
        return n;
```

```
    }
```

```
    //方法2：使用Random.nextInt()
```

```
    public static int rnd2( int min, int max ) {
```

```
        Random r = new Random(); //创建Random对象 ← import java.util.Random
```

```
        int n= min + r.nextInt(max-min+1); // [min,max]
```

```
        return n;
```

```
    }
```

```
//主程序
```

```
public static void main(String[] args) {
```

```
    int min=10;
```

```
    int max=20;
```

```
    for ( ; ; ) {
```

```
        int x = rnd(min, max);
```

```
        System.out.println("随机数=" + x);
```

```
        if (x == min || x == max)
```

```
            break;
```

```
    }
```

```
}
```

**一个运行结果**

```
随机数=18
随机数=11
随机数=12
随机数=20
```

Random.nextInt(): 产生随机整数  
Random.nextInt(t): 产生 [0,t) 随机整数

**【返回】**

## 4. 素数

Ex\_4.java

```
public class Ex_4 {  
    //素数判定  
    public static boolean isPrime(int n) {  
        if (n <= 1) {    //0,1,负数都不是素数  
            return false;  
        } else {  
            for (int i = 2; i <= Math.sqrt(n); i++) {  
                if (n % i == 0)  
                    return false;  
            }  
        }  
        return true;  
    }  
}
```

//主程序

```
public static void main(String[] args) {  
    int count = 0;  
    for (int n = 1; n <= 100; n++) {  
        if ( isPrime(n) ) {  
            count++;  
            System.out.println(n);  
        }  
    }  
    System.out.println("素数一共" + count + "个");  
}
```

[【返回】](#)

## 5. 水仙花数

例如：153 =  $1^3 + 5^3 + 3^3$  就是一个水仙花数

Ex\_5.java

```
public class Ex_5 {
```

```
    //水仙花判定
```

```
    public static boolean isFlower(int n) {
```

```
        int ge, shi, bai, sum = 0;
```

```
        ge = n % 10;
```

```
        shi = n % 100 / 10;
```

```
        bai = n / 100;
```

```
        sum = ge*ge*ge + shi*shi*shi + bai*bai*bai;
```

```
        if (sum == n)
```

```
            return true;
```

```
        else
```

```
            return false;
```

```
    }
```

分解各位数字

```
    //主程序
```

```
    public static void main(String[] args) {
```

```
        for (int n = 100; n < 1000; n++) {
```

```
            if ( isFlower(n) )
```

```
                System.out.println(n + "是一个水仙花数");
```

```
        }
```

```
    }
```

```
}
```

运行结果

153是一个水仙花数

370是一个水仙花数

371是一个水仙花数

407是一个水仙花数

[【返回】](#)

## 6. 计算圆周率 $\pi$

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots (-1)^{m+1} \frac{1}{2m-1} + \cdots$$

Ex\_6.java

```
public class Ex_6 {  
    //计算圆周率  
    public static double getPI() {  
        double pi_quarter = 0.0;  
        double flag = 1.0;  
        for (int i = 1; i < 10000000; i += 2, flag = -flag) {  
            pi_quarter += flag / i;  
        }  
        return 4*pi_quarter;  
    }  
    //主程序  
    public static void main(String[] args) {  
        System.out.println(getPI());  
    }  
}
```

运行结果

3.1415924535897797

[【返回】](#)

## 7. 递归：求 $2+4+6+\dots+100$

分析：  $S = 100 + 98 + 96 + \dots + 2$

把每个值看成是一个函数返回的值

$$S(100) = 100 + S(98)$$

$$= 100 + 98 + S(96)$$

$$= 100 + 98 + 96 + S(94) \dots$$

$$= 100 + 98 + 96 + 94 + \dots + S(2)$$

$$= 100 + 98 + 96 + 94 + \dots + 2$$

得到递归公式：

$$S(n) = \begin{cases} 2 & n=2 \\ n + S(n-2) & n>2 \end{cases}$$

## 递归程序

```
public class Ex_7 {  
    //递归求和  
    public static int S(int n) {  
        if (n == 2)  
            return 2;  
        else  
            return n + S(n - 2);  
    }  
  
    //主程序  
    public static void main(String[] args) {  
        System.out.print( S(100) );  
    }  
}
```

[【返回】](#)

## 8. 最小值和最大值

Ex\_6.java

```
public class Ex_8 {  
    //主程序  
    public static void main(String[] args) {  
        System.out.println("请输入两个整数:");  
        Scanner sc = new Scanner(System.in);  
        int a = sc.nextInt();  
        int b = sc.nextInt();  
        System.out.println(a + "和" + b + "的最小值是: " + Math.min(a, b));  
        System.out.println(a + "和" + b + "的最大值是: " + Math.max(a, b));  
    }  
}
```

运行结果

```
请输入两个整数:  
12  
-34  
12和-34的最小值是: -34  
12和-34的最大值是: 12
```

# Math类常用属性和方法

属性或方法	说明
PI	π值
E	e值
abs()	求绝对值
sqrt()	求平方根
pow(double a, double n)	a的n次幂
exp(double n)	e的n次幂
min(double d1, double d2)	返回d1与d2中的小者
max(double d1, double d2)	返回d1与d2中的大者
ceil(double a)	返回大于或等于 a 的最小整数
floor(double a)	返回小于或等于 a 的最大整数
round(double d)	四舍五入取整(将原来的数字加上 0.5 后再向下取整)
random()	返回 [ 0,1) 的随机数

【完】