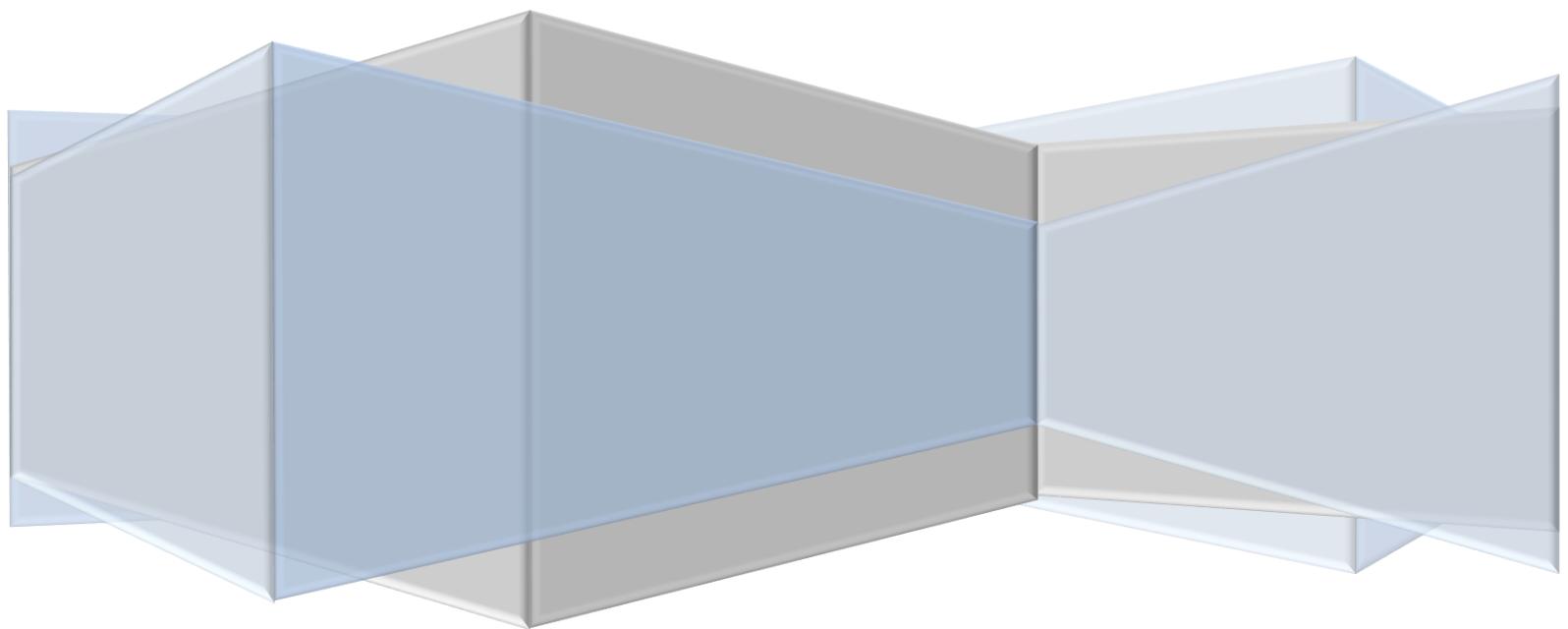


本人编写仅供学习，请勿传播



# JAVA 语言开发基础

Java 小白著 2017



## 目录

第一章、Java 语言开发基础(变量常量、运算符)	4
1.1 Java 语言发展史	4
1.2 Java 语言特性	4
1.3 JDK 和 JRE 的解释	5
1.4 JAVA 环境变量配置	5
1.5 第一个 JAVA 程序 (HelloWorld)	6
1.5.1 程序编写要求	6
1.5.2 程序执行过程	7
1.5.3 代码注释	7
1.6 关键字概述	8
1.6.1 关键字概述	8
1.6.2 关键字特点	8
1.6.3 关键字注意事项	8
1.6.4 常见关键字	8
1.6.5 标识符	9
1.7 常量和变量	9
1.7.1 常量	9
1.7.2 变量	10
1.7.3 成员变量和局部变量的区别	11
1.8 常用数据类型和运算符	11
1.8.1 常用数据类型及取值范围	11
1.8.2 数据类型转换	12
1.8.3 算术运算符	12
1.8.4 赋值运算符	13
1.8.5 关系运算符	13
1.8.6 逻辑运算符	13
1.8.7 位运算符	14
1.8.8 三目运算符	14
1.8.9 常用运算符的优先级顺序	15

1.9 流程控制语句 .....	16
1.9.1 顺序结构 .....	16
1.9.2 选择结构 .....	17
1.9.3 循环结构 .....	22
1.9.4 跳转控制语句 .....	26
<b>第二章、类与对象（抽象、封装） .....</b>	<b>28</b>
2.1 抽象的概念 .....	28
2.2 封装的概念 .....	29
2.3 类与对象 .....	31
2.3.1 类的设计 .....	31
2.3.2 创建对象和构造方法 .....	32
2.3.3 域和方法（关键字 <code>static</code> 和 <code>final</code> ） .....	33
2.3.4 静态方法（变量）和非静态方法（变量）的区别 .....	33
2.4 继承与多态 .....	34
2.4.1 继承的概念 .....	34
2.4.2 继承的语法 .....	34
2.4.3 多态的概念 .....	35
2.4.4 方法重载和方法覆盖（重写） .....	37
2.4.5 关键字 <code>this</code> 和关键字 <code>super</code> .....	39
2.4.6 构造方法的继承和重载 .....	41
2.5 抽象类与抽象方法 .....	43
2.5.1 抽象类的概念 .....	43
<b>第三章、JAVA 常用工具类 .....</b>	<b>44</b>
3.1 Object 类 .....	44
3.2 基本数据对象包装类 .....	45
3.3 Java 的字符串 .....	47

---

3.3.1 String 类.....	47
3.3.2 StringBuffer.....	52
3.3.4 StringBuilder.....	53
<b>3.4 Math 类.....</b>	<b>53</b>
3.4.1 三角函数方法.....	54
3.4.2 指数函数方法.....	54
3.4.3 取整函数方法.....	55
3.4.4 取最大值、最小值、绝对值函数.....	55
3.4.5 Math 中定义的常量.....	55
3.4.6 Math.random()函数.....	56
3.4.7 大数据运算.....	56
<b>3.5 Date 类.....</b>	<b>59</b>
3.5.1 构造方法.....	59
3.5.2 常用的方法（未过时）.....	60
3.5.3 日期类型的其他类.....	60
<b>第四章、JavaWeb 基础.....</b>	<b>62</b>
<b>4.1 Web 运行环境 Tomcat 配置.....</b>	<b>62</b>
4.1.1 名词解释.....	62
4.1.2 目录结构及用途.....	62
4.1.3 JAR 存放位置.....	63
4.1.4 Tomcat 的配置文件.....	64
4.1.5 环境搭建.....	65
<b>4.2 JSP 常用内置对象.....</b>	<b>66</b>

# 第一章、Java 语言开发基础(变量常量、运算符)

## 1.1 Java 语言发展史

Java 之父——詹姆斯·高斯林 (James Gosling)

1977 年获得了加拿大卡尔加里大学计算机科学学士学位, 1983 年获得了美国卡内基梅隆大学计算机科学博士学位, 毕业后到 IBM 工作, 设计 IBM 第一代工作站 NeWS 系统, 但不受重视。后来转至 Sun 公司, 1990 年, 与 Patrick, Naughton 和 Mike Sheridan 等人合作“绿色计划”, 后来发展一套语言叫做“Oak”, 后改名为 Java。

## 1.2 Java 语言特性

简单性、面向对象、可移植性 (跨平台)、分布式、多线程、动态性、健壮性、安全性  
可移植性。

### 什么是跨平台性?

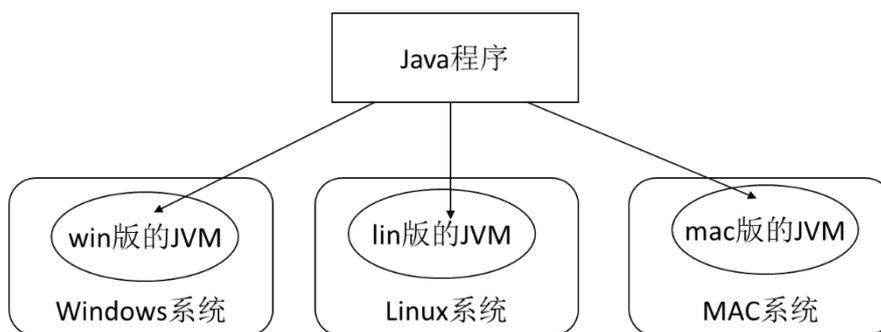
通过 Java 语言编写的应用程序在不同的系统平台上都可以运行。

### 原理是什么?

只要在需要运行 java 应用程序的操作系统上,

先安装一个 Java 虚拟机(JVM Java Virtual Machine)即可。

由 JVM 来负责 Java 程序在该系统中的运行。



因为有了 JVM, 所以同一个 Java 程序在三个不同的操作系统中都可以执行。这样就实现了 Java 程序的可移植性。也称为 Java 具有良好的跨平台性。

### 1.3 JDK 和 JRE 的解释

JRE(Java Runtime Environment Java 运行环境)

包括 Java 虚拟机(JVM Java Virtual Machine)和 Java 程序所需的核心类库等, 如果想要运行一个开发好的 Java 程序, 计算机中只需要安装 JRE 即可。

JDK(Java Development Kit Java 开发工具包)

JDK 是提供给 Java 开发人员使用的, 其中包含了 java 的开发工具, 也包括了 JRE。所以安装了 JDK, 就不用单独安装 JRE 了。

其中的开发工具: 编译工具(javac.exe) 执行工具(java.exe) 打包工具(jar.exe)等

**简单而言: 使用 JDK 开发完成的 java 程序, 交给 JRE 去运行。(jvm 保证跨平台)**

### 1.4 JAVA 环境变量配置

三个环境变量

**path:**去哪里找编译或运行等工具(必须设置)

**JAVA\_HOME:**JDK 的安装目录

**classpath:**去哪里找需要运行的 class 文件

通过 javac 命令验证

常见问题：“javac 不是内部或者外部命令”，原因是 path 配置错误

避免 C:\WINDOWS\system32 出现所有名字以 java 开头的文件，有，删之；

注:jdk 不要安装在带中文的路径下,最好也是不包含空格字符的英文路径;

在系统变量 Path 中添加如下变量%JAVA\_HOME%\bin;

## 1.5 第一个 JAVA 程序（HelloWorld）

### 1.5.1 程序编写要求

首先定义一个类 class 类名

在类定义后加上一对大括号{}

在大括号中间添加一个主(main)方法/函数

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

在主方法的大括号中间添加一行输出语句

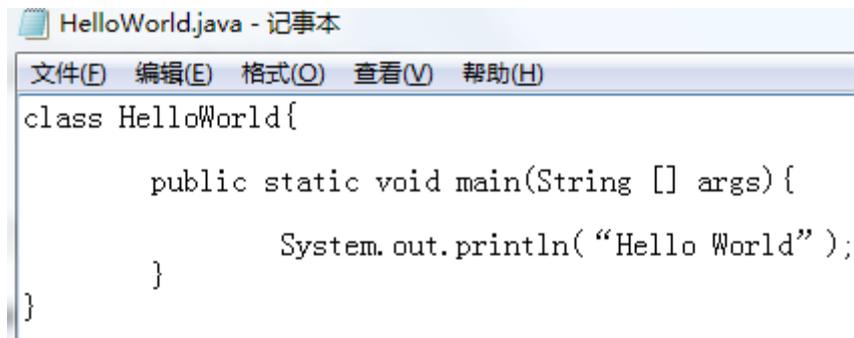
```
System.out.println( "hello world" );
```

程序演示:

```
class HelloWorld{  
  
    public static void main(String [] args){  
  
        System.out.println( "Hello World" );  
  
    }  
  
}
```

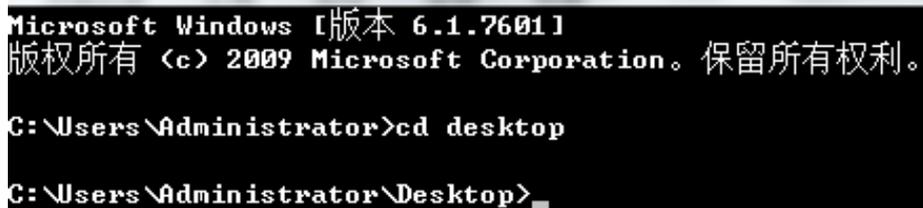
## 1.5.2 程序执行过程

1 编写好代码（注意类名和文件名最好保持一致）



```
>HelloWorld.java - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
class HelloWorld{
    public static void main(String [] args){
        System.out.println(“Hello World”);
    }
}
```

1. 在命令（cmd）行进入文件所属的文件夹（cd 命令可在当前目录下进行切换）。



```
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>cd desktop

C:\Users\Administrator\Desktop>_
```

2. 使用 javac 文件名.java 进行编译，如果提示错误，更改错误后重新进行编译



```
C:\Users\Administrator\Desktop>javac HelloWorld.java
```

3. 使用 java 文件名进行执行，查看程序运行结果



```
C:\Users\Administrator\Desktop>java HelloWorld
Hello World
```

## 1.5.3 代码注释

### 注释概述

用于解释说明程序的文字

### Java 中注释分类格式

单行注释 格式：//注释文字

多行注释 格式：/\* 注释文字 \*/

**文档注释** 格式: `/** 注释文字 */`

注释是一个程序员必须要具有的良好编程习惯。初学者编写程序可以养成习惯: 先写注释再写代码。将自己的思想通过注释先整理出来, 在用代码去体现。因为代码仅仅是思想的一种体现形式而已。解释说明程序, 提高程序的阅读性

## 1.6 关键字概述

### 1.6.1 关键字概述

被 Java 语言赋予特定含义的单词

### 1.6.2 关键字特点

组成关键字的字母全部小写

### 1.6.3 关键字注意事项

`goto` 和 `const` 作为保留字存在, 目前并不使用

类似 Notepad++ 这样的高级记事本, 针对关键字有特殊的颜色标记, 非常直观

### 1.6.4 常见关键字

用于定义数据类型的关键字				
class	interface	byte	short	int
long	float	double	char	boolean
void				
用于定义数据类型值的关键词				
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				
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

## 1.6.5 标识符

### 标识符概述

就是给类,接口,方法,变量等起名字时使用的字符序列

### 组成规则

英文大小写字母、数字字符、\$和\_

### 注意事项:

不能以数字开头

不能是 Java 中的关键字

区分大小写

## 1.7 常量和变量

### 1.7.1 常量

#### ➤ 常量概述

在程序执行的过程中其值不可以发生改变

#### ➤ Java 中常量分类

- 字面值常量
  - 自定义常量(面向对象部分讲)
  - 字符串常量 用双引号括起来的内容
  - 整数常量 所有整数  
12,23
  - 小数常量 所有小数  
12.34,56.78
  - 字符常量 用单引号括起来的内容  
'a' ; 'A' ; '0'
  - 布尔常量 较为特有, 只有 true 和 false
4. 空常量 null(数组部分讲解)

## 1.7.2 变量

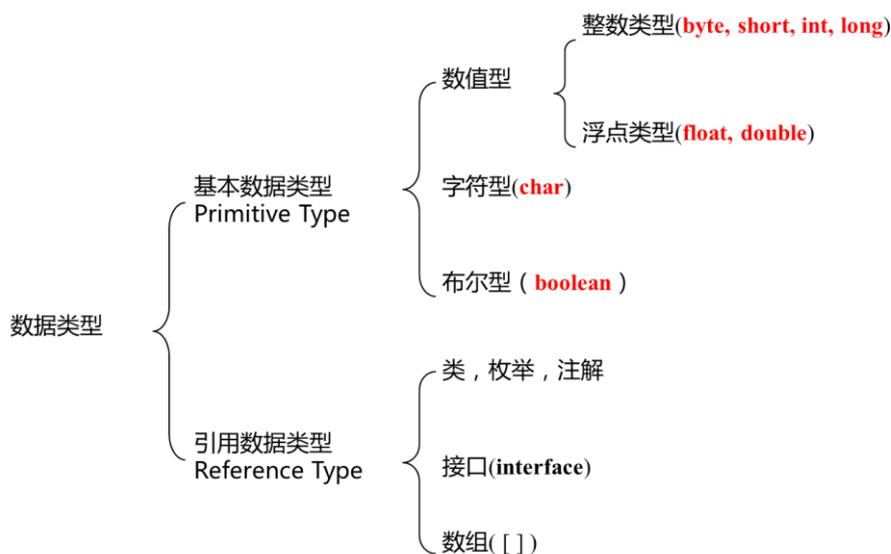
- 在程序执行的过程中, 在某个范围内其值可以发生改变的量  
(理解: 如同数学中的未知数)
- 变量定义格式  
**数据类型 变量名 = 初始化值;**  
注意: **格式是固定的**, 记住格式, 以不变应万变
- 作用范围: **定义开始到定义它的代码块结束;**  
注意: 同一范围内, **不允许多个局部变量命名冲突**  
定义开始到定义它的代码块结束

### 1.7.3 成员变量和局部变量的区别

- a) 局部变量: 不是声明在类体括号里面的变量;
- b) 局部变量使用前**必须**初始化值;
- c) 局部变量没有默认初始化值;
- d) 局部变量的作用域是从定义开始到定义它的代码块结束;
- e) 成员变量:在方法体外,类体内声明的变量, 又称字段(Field)或全局变量;
- f) 成员变量的作用域是整个类中;

## 1.8 常用数据类型和运算符

### 1.8.1 常用数据类型及取值范围



#### ➤ 数据类型取值范围

序号	数据类型	大小/位(B)	数据范围
1	byte	8	[-128,127]
2	short	16	[-2 <sup>15</sup> ,2 <sup>15</sup> -1]

3	int	32	$[-2^{31}, 2^{31}-1]$
4	long	64	$[-2^{63}, 2^{63}-1]$
5	char	16	$[0, 2^{16}-1]$
6	float	32	$[-3.4E38(-3.4*10^{38}),$ $3.4E38(-1.7*10^{30})];$
7	double	64	$[-1.7E308(-1.7*10^{308}),$ $1.7E308(-1.7*10^{308})];$
8	boolean	1	true / false

### 1.8.2 数据类型转换

boolean 类型不能转换为其他的数据类型

默认转换

byte, short, char—int—long—float—double

byte, short, char 相互之间补转换，他们参与运算首先转换为 int 类型

强制转换

目标类型 变量名=(目标类型)(被转换的数据);

### 1.8.3 算数运算符

+ - \* / % (取余) ++(自加) -- (自减) +

注意：+号在字符串中起到一个拼接的作用

## 1.8.4 赋值运算符

=(赋值) +=(加等) -=(减等) \*=(乘等) /=(除等) %=(模等)

举例: `a+=3` ——>`a=a+3;`

## 1.8.5 关系运算符

==(相等于) !=(不等) >=(大于等于)

<=(小于等于) >(大于) <(小于) instanceof (检查是否是某一类的对象)

注 1: 比较运算符的结果都是 boolean 型, 也就是要么是 true, 要么是 false。

注 2: 比较运算符 “==” 不能误写成 “=” 。

举例 `instanceof` `if(num instanceof Integer)`

## 1.8.6 逻辑运算符

&(逻辑与) | (逻辑或) !(逻辑非)

&&(短路与) || (短路或) ^(异或)

逻辑运算符注意: 在 Java 中不可以写成 `3<x<6`, 应该写成 `x>3 & x<6` 。

“&” 和 “&&” 的区别:

单&时, 左边无论真假, 右边都进行运算;

双&时, 如果左边为真, 右边参与运算, 如果左边为假, 那么右边不参与运算。

“|” 和 “||” 的区别同理, 双或时, 左边为真, 右边不参与运算。

异或(^)与或(|)的不同之处是: 当左右都为 true 时, 结果为 false。(相同为假, 不同为真)

### 1.8.7 位运算符

<<(逻辑左移)    >>(逻辑右移)    >>(无符号右移)

2 (0010)<<2=8(1000);左移两位

4(0100)>>2=1(0001);右移两位

注意: >>>无论最高位是 0 还是 1, 空缺位都用 0 来补

### 1.8.8 三目运算符

格式: (关系表达式)?表达式 1: 表达式 2;

如果条件为 true, 运算后的结果是表达式 1;

如果条件为 false, 运算后的结果是表达式 2;

示例:

获取两个数中大数。

```
int x=3,y=4,z;
```

```
z = (x>y)?x:y;//z 变量存储的就是两个数的大数。
```

## 1.8.9 常用运算符的优先级顺序

运算符的优先级（从高到低）

优先级	描述	运算符
1	括号	()、[]
2	正负号	+, -
3	自增自减, 非	++, --, !
4	乘除, 取余	*, /, %
5	加减	+, -
6	移位运算	<<, >>, >>>
7	大小关系	>, >=, <, <=
8	相等关系	==, !=
9	按位与	&
10	按位异或	^
11	按位或	
12	逻辑与	&&
13	逻辑或	
14	条件运算	?:
15	赋值运算	=, +=, -=, *=, /=, %=
16	位赋值运算	&=,  =, <<=, >>=, >>>=

### 课后习题:

#### 面试题

```
byte b1=3,b2=4,b;
```

```
b=b1+b2;
```

```
b=3+4;
```

哪句是编译失败的呢? 为什么呢?

#### 思考题

byte b = 130;有没有问题?如果我想让赋值正确, 可以怎么做?结果是多少呢?

#### 请写出下列程序结果

```
System.out.println( 'a' );
```

```
System.out.println( 'a' +1);
```

```
System.out.println( "hello" + ' a' +1);
```

```
System.out.println( 'a' +1+" hello" );
```

```
System.out.println( "5+5=" +5+5);
```

```
System.out.println(5+5+" =5+5" );
```

如下操作写出结果

```
int a,b; a = b = 10;
```

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

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

```
int a = 10; a += 20;
```

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

练习题

```
short s=1, s = s+1; short s=1, s+=1;
```

上面两个代码有没有问题，如果有，那里有问题

## 1.9 流程控制语句

➤ 在一个程序执行的过程中，各条语句的执行顺序对程序的结果是有直接影响的。也就是说程序的流程对运行结果有直接的影响。所以，我们必须清楚每条语句的执行流程。而且，很多时候我们要通过控制语句的执行顺序来实现我们要完成的功能。

➤ 流程控制语句分类

顺序结构

选择结构

循环结构

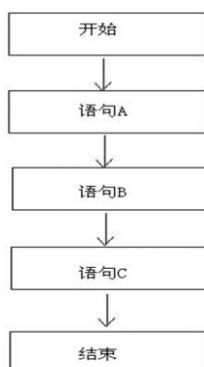
### 1.9.1 顺序结构

➤ 顺序结构概述

是程序中最简单最基本的流程控制，没有特定的语法结构，按照代码的先后顺序，依次执行，程序中大多数的代码都是这样执行的。

总的来说：写在前面的先执行，写在后面的后执行

➤ 顺序结构图



## 1.9.2 选择结构

➤ 选择结构也被称为分支结构。

选择结构有特定的语法规则，代码要执行具体的逻辑运算进行判断，逻辑运算的结果有两个，所以产生选择，按照不同的选择执行不同的代码。

➤ Java 语言提供了两种选择结构语句

if 语句

switch 语句

➤ 选择结构(if)

➤ **if 语句有三种格式**

**1. if 语句第一种格式:**

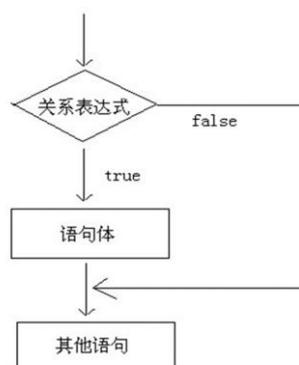
```
if(关系表达式) {  
    语句体
```

```
}
```

## 执行流程

首先判断关系表达式看其结果是 true 还是 false,如果是 true 就执行语句体,如果是 false 就不执行语句体

## 流程图



## 注意事项

关系表达式无论简单还是复杂,结果必须是 boolean 类型

if 语句控制的语句体如果是一条语句,大括号可以省略;如果是多条语句,就不能省略。

建议永远不要省略。

一般来说:有左大括号就没有分号,有分号就没有左大括号

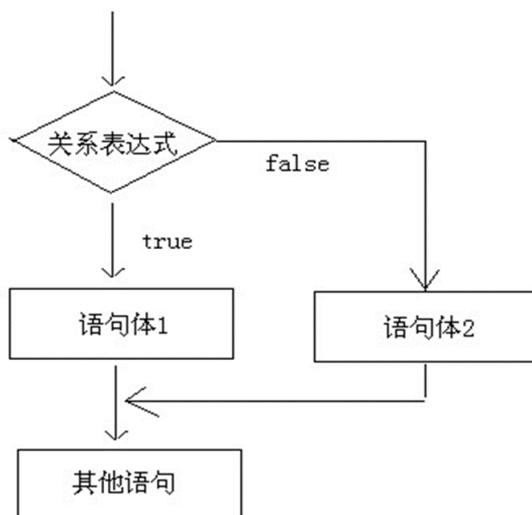
## 2. if 语句第二种格式

```
if(关系表达式) {  
    语句体 1;  
}  
else {  
    语句体 2;  
}
```

## 执行流程

首先判断关系表达式看其结果是 true 还是 false,如果是 true 就执行语句体 1,如果是 false 就执行语句体 2。

**流程图:**



**注意事项:**

我们前面讲解过三元运算符,它根据比较判断后,给出的也是两个结果,所以,这种情况和 if 语句的第二种格式很相似,他们在某些情况下应该是可以相互转换的。

if 语句第二种格式和三元运算符的比较:三元运算符的操作都可以使用 if 语句改进,反之不成立,什么时候不成立呢?

当 if 语句控制的语句体是一条输出语句的时候,就不成立。因为三元运算符是一个运算符,必须要求有一个结果返回。而输出语句却不能作为一个返回结果。

### 3. if 语句的第三种格式

```
if(关系表达式 1) {  
    语句体 1;  
}else if (关系表达式 2) {  
    语句体 2;
```

```
    }  
    ...  
    else {  
        语句体 n+1;  
    }
```

### 执行流程

首先判断关系表达式 1 看其结果是 true 还是 false, 如果是 true 就执行语句体 1, 如果是 false 就继续判断关系表达式 2 看其结果是 true 还是 false, 如果是 true 就执行语句体 2, 如果是 false 就继续判断关系表达式, ...看其结果是 true 还是 false..., 如果没有任何关系表达式为 true, 就执行语句体 n+1。

### ➤ 选择结构(switch 语句)

switch 语句格式:

```
switch(表达式) {  
    case 值 1:  
        语句体 1;  
        break;  
    case 值 2:  
        语句体 2;  
        break;  
    ...  
    default:  
        语句体 n+1;
```

```
break;  
  
}
```

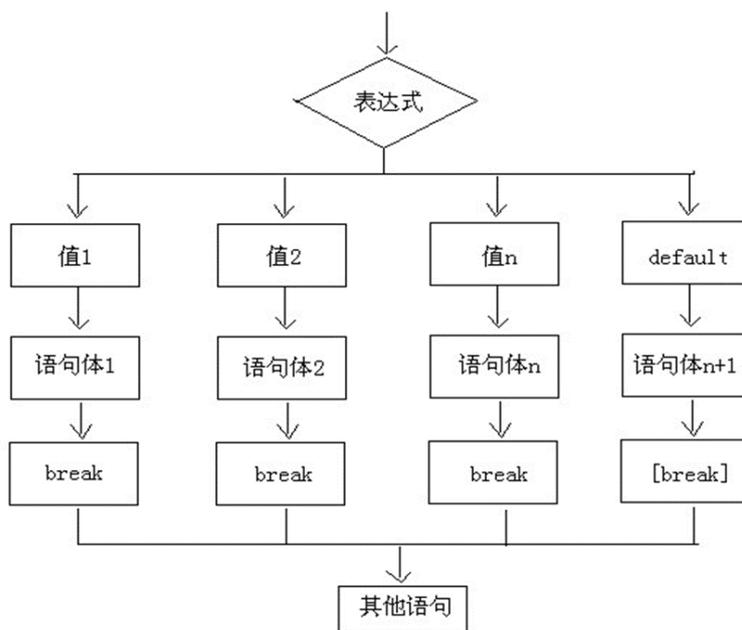
### 格式解释

switch 表示这是 switch 语句，表达式的取值：byte,short,int,char，JDK5 以后可以枚举，JDK7 以后可以是 String，case 后面跟的是要和表达式进行比较的值，语句体部分可以是一条或多条语句，break 表示中断，结束的意思，可以结束 switch 语句，default 语句表示所有情况都不匹配的时候，就执行该处的内容，和 if 语句的 else 相似。

### 执行流程

首先计算出表达式的值，其次，和 case 依次比较，一旦有对应的值，就会执行相应的语句，在执行的过程中，遇到 break 就会结束。最后，如果所有的 case 都和表达式的值不匹配，就会执行 default 语句体部分，然后程序结束掉。

### 执行流程图



### 注意事项

- case 后面只能是常量，不能是变量，而且，多个 case 后面的值不能出现相同的，
- default 一般不建议省略。除非判断的值是固定的。
- break 一般不建议省略。否则结果可能不是你想要的
- default 的位置可以出现在 switch 语句任意位置。
- switch 语句的结束条件：遇到 break 执行到程序的末尾

### 问题

在做判断的时候，我们有两种选择，if 语句和 switch 语句，那么，我们到底该如何选择使用那种语句呢？

if 语句使用场景：针对结果是 boolean 类型的判断、针对一个范围的判断、针对几个常量值的判断。switch 语句使用场景：针对几个常量值的判断

## 1.9.3 循环结构

### ➤ 循环结构介绍

循环语句可以在满足循环条件的情况下，反复执行某一段代码，这段被重复执行的代码被称为循环体语句，当反复执行这个循环体时，需要在合适的时候把循环判断条件修改为 false，从而结束循环，否则循环将一直执行下去，形成死循环。

### ➤ 循环语句的组成

#### 初始化语句：

一条或者多条语句，这些语句完成一些初始化操作。

#### 判断条件语句：

这是一个 boolean 表达式，这个表达式循环体。能决定是否执行

#### 循环体语句：

这个部分是循环体语句，也就是我们要多次做的事情。

### 控制条件语句：

这个部分在一次循环体语句结束后，下一次循环判断条件执行前执行。通过用于控制循环条件中的变量，使得循环在合适的时候结束。

## 1. 循环语句 for;

### for 循环语句格式：

```
for(初始化语句;判断条件语句;控制条件语句) {  
    循环体语句;  
}
```

### 执行流程

A:执行初始化语句

B:执行判断条件语句，看其结果是 true 还是 false,如果是 false，循环结束。如果是 true，继续执行。

C:执行循环体语句

D:执行控制条件语句

E:回到 B 继续

### 注意事项

判断条件语句的结果是一个 boolean 类型

循环体语句如果是一条语句，大括号可以省略；如果是多条语句，大括号不能省略。建议永远不要省略。

一般来说：有左大括号就没有分号，有分号就没有左大括号

## 2. 循环语句 while

### 基本格式

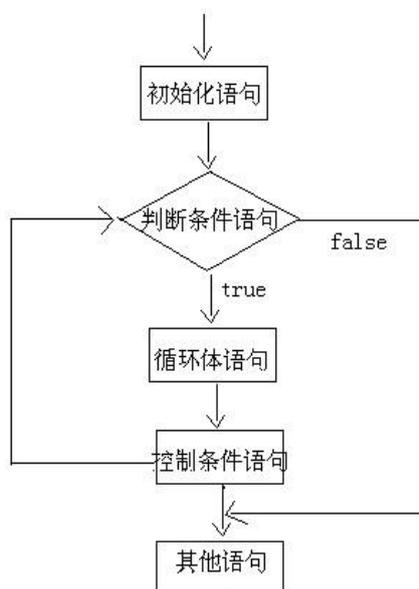
```
while(判断条件语句) {  
    循环体语句;  
}
```

### 扩展格式

初始化语句;

```
while(判断条件语句) {  
    循环体语句;  
    控制条件语句;  
}
```

While 语句格式图:



## 3. 循环结构 do...while ()

### 基本格式

```
do {  
    循环体语句;
```

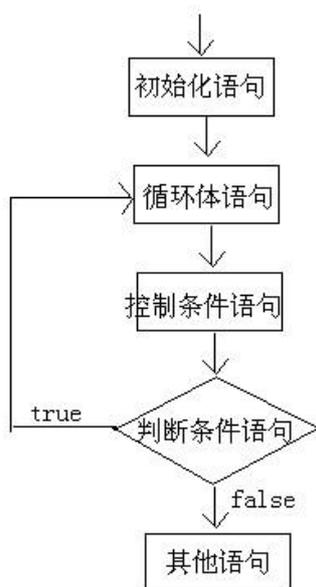
```
}while(判断条件语句);
```

### 扩展格式

```
初始化语句;
```

```
do {  
    循环体语句;  
    控制条件语句;  
} while(判断条件语句);
```

### do...while 循环语句格式图



注意：无论条件成立与否，循环体会至少执行一次。

## 4. for 循环和 while 循环的区别

for 循环语句和 while 循环语句可以等价转换，但还是有些小区别的

### 使用区别：

控制条件语句所控制的那个变量，在 for 循环结束后，就不能再被访问到了，而 while 循环结束还可以继续使用，如果你想继续使用，就用 while，否则推荐使用 for。原因是 for 循环结束，该变量就从内存中消失，能够提高内存的使用效率。

三种循环语句其实都可以完成一样的功能,也就是说可以等价转换,但还是有小区别的:

do...while 循环至少会执行一次循环体。

for 循环和 while 循环只有在条件成立的时候才会去执行循环体

### 场景区别:

for 循环适合针对一个范围判断进行操作, while 循环适合判断次数不明确操作

### 注意事项

写程序优先考虑 for 循环,再考虑 while 循环,最后考虑 do...while 循环。

如下代码是死循环

```
while(true){}
```

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

## 1.9.4 跳转控制语句

### ➤ 解释:

在某个循环到某一步的时候就结束,现在就做不了这件事情。为了弥补这个缺陷,Java 就提供了 break, continue 和 return 来实现控制语句的跳转和中断。

break 中断当前循环,终止循环程序

continue 继续,终止本层循环,不影响下次循环

return 返回一个值,结束一个方法

### 1. 跳转控制语句 break

#### break 的使用场景:

在选择结构 switch 语句中

在循环语句中

**break 的作用:**

终止该层循环;

**2. 跳转控制语句(continue)****continue 的使用场景:**

在循环语句中

**continue 的作用:**

跳过该层循环

break 退出当前循环, continue 退出本次循环,继续下一次循

**3. 跳转控制语句(return)**

return, 更常用的功能是结束一个方法, 也就是退关键字不是为了跳转出循环体出一个方法。跳转到上层调用的方法。这个在方法的使用那里会在详细的讲解。

**🌈 循环语句练习题**

请在控制台输出数据 1-10

请在控制台输出数据 10-1

求出 1-10 之间数据之和

求出 1-100 之间偶数和

求出 1-100 之间奇数和

求 5 的阶乘

在控制台输出所有的“水仙花数”

统计“水仙花数”共有多少个 (100~1000 之间)

## 第二章、类与对象（抽象、封装）

**面向对象编程** (Object Oriented Programming), 简称 **OOP**, 是一种新兴的程序设计方法, 其思想是从现实世界中客观存在的事物 (对象) 出发, 来进行构造软件系统, 并且在编写的过程中尽量运用人的思维方式。

**世间一切万物皆对象**。类与对象之间的关系类似于**铸件**和**模具**的关系, 类是某一类对象的封装体。类的**实例化**的结果就是对象。

**抽象、封装、继承和多态**是 java 语言中面向对象的基本特征。

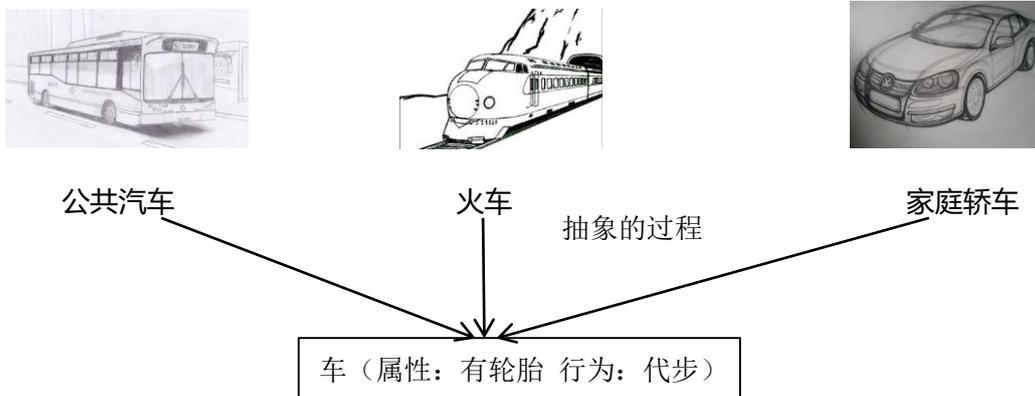
### 2.1 抽象的概念

将一类属性以及行为相似事物来模糊的归为一类的过程, 就称之为一个抽象的过程。抽象能够界定出对象的状态特征 (属性) 和基本行为, 但抽象并不是研究所有目标对象的全部过程, 而是选择其中最基本的特征, 舍弃掉与研究对象无关系或者关系不大的方面。

#### ➤ 解释

矩形类包含的属性有: 长度和宽度, 那么这两个属性就是对象所共有的状态特征, 而矩形的面积是一些共有的行为。

#### ➤ 抽象的过程



## 2.2 封装的概念

封装的目的是形成一个具有特殊功能和指定方法的类, 封装是实现抽象的必经之路, 所谓封装就是把对象的全部属性和全部功能结合在一起, 包装成一个不可分割的单元(对象), 对象就是一组域和相关方法的封装, **属性+功能=封装**。

### ➤ 封装的实现机制

1. 设置域和成员方法的访问权限。
2. 提供一个供其它类引用的同一方法接口。
3. 其他对象不能直接修改对象所拥有的域和方法。

### ➤ 访问权限的四种类型:

1. 公共 (public)
2. 保护 (protected)
3. 缺省 (pack-age)
4. 私有 (private)

**公共**的访问权限的域(属性)和成员方法, 可以供其他类进行使用, 而不具有信息隐藏的功能, 并且访问权限几乎不存在。

**保护**的访问权限定义的域和成员方法, 可以为其类本身或者同属一个包的其他类所访问, 也可以被该类的子类所访问, 即便子类与该类不属于同一个包。

**缺省**的访问权限又称为包访问权限, 这类域和成员方法只对于所属于同一个包中的其它类可以访问, 而包外的类则不能直接访问它们。

**私有**的访问权限的域和成员方法的访问机制最为严格, 只能在本类的类体中使用, 如果有的成员不想让其他类进行访问, 也可将成员或者属性设置为 private 访问权限。信息隐藏的主要手段就是设置隐藏信息的访问权限为 private。

### ➤ 程序举例

定义一个学生类，类封装的要求为：将学生的姓名、性别、年龄等属性设置为私有，该类提供的方法都设置为共有方法。

```
class Student{

    private String name;//姓名域

    private String sex;//性别域

    private int age;//年龄域

    //设置学生姓名的方法

    public void setName(String stuName){

        name= stuName;}

    //获取学生姓名的方法

    public String getName(){

        return name;

        } .....

    //设置学生 a 年龄的方法

    public void setAge(int stuAge){

        age = stuAge;    }

    //获取学生的年龄的方法

    public int getAge(){

        return age;

    }

}
```

## 2.3 类与对象

### 2.3.1 类的设计

类是组成 Java 程序的结构部件，也是面向对象编程的基本元素，它定义了类对象的结构和功能，封装类对象的状态和方法，是类对象的模型。

类的实现包括两个部分：类声明和类体定义：

➤ 类声明的完整格式：

```
[public] [abstract] [final] class<类名> [extends<父类名>] [implements<接口列表>]
```

- 1) class 为定义类的关键字。
- 2) 修饰符 public、abstract、final 说明了类的特性。public 关键字声明当前类为公共类，可以被其它类所访问。Abstract 关键字声明当前类为抽象类，抽象类只能被继承，不能被实例化，final 关键字声明该类为最终类，该类不能被继承。
- 3) 注意:abstract 和 final 不能同时出现。
- 4) 类名为类的名称，它应该是一个合法字符，要求类名首字母大写，不能以阿拉伯数字开头，类名要求写成顾名思义的，不要使用无意义的类名来命名。
- 5) implements 关键字用来实现接口，接口可以有多个，各接口之间以逗号分隔:
- 6) 类体的定义格式为：

```
{  
  
    //域的定义;  
  
    //构造函数的定义;  
  
    //成员方法的定义;  
  
}
```

## 2.3.2 创建对象和构造方法

### ➤ 创建对象

从一个类生成若干个实例的过程称为创建对象, 创建对象包括对象声明和对象初始化两个步骤。 声明一个对象的格式为:

1. <类名> 对象名 例如: 声明 Student 一个对象 student 的语句是: Student student
2. 用 new 关键字给对象分配内存空间, 格式为: <对象名>=new 类名 (实参列表)
3. 创建对象格式: 类名 对象名=new 类名 () 参数是否传递请参考构造方法

### ➤ 构造方法

创建对象时, 需要调用类的构造方法, **构造方法是用来创建实例对象**, 初始化对象值域的特殊方法, 构造方法名与类名相同, 在创建空间对象后, 系统根据参数, 调用相对应的构造方法。 **构造方法主要功能是为数据成员赋值**, 每一个类**至少**有一个构造方法, 否则无法创建对象, 如果没有创建构造方法, 系统会**默认创造**一个无参数的构造方法。

注意:

1. 如果没有创建构造方法, 系统会默认创造一个无参数的构造方法
2. 构造方法没有返回值, 也不允许 void 返回类型的声明。
3. 构造方法的名称必须与类名相同。
4. 构造方法不允许用户直接调用, 必需使用 new 关键字创建对象时, 由系统自动调用。
5. 一个类可以创建多个构造方法, 构造方法之间, 通过不同的参数来区分。
6. 构造方法可以继承, 子类可以继承父类的构造方法。
7. 构造方法的目的是在创建对象时初始化类的内部状态。
8. 构造方法在创建时最长使用 public 来声明, 极少会使用 private 等 (特殊地方使用)
9. 构造方法之间使用 this 关键字来调用。(参考 this 关键字)

### 2.3.3 域和方法（关键字 `static` 和 `final`）

- 域的定义：称为属性或成员变量，在类中还可以定义成员变量。

域(成员变量)的定义格式：

```
[public | protected | private] [static] [final] <域名>
```

- 成员方法的定义格式：

```
[public | protected | private] [static] [final | abstract] <返回类型> <方法名> ([参数列表]) [throws <异常列表>]
```

- 域的修饰符（常用）

1. `static`：用于声明静态域的修饰符，静态域是与实例域相对的概念。
2. `final`：用于声明常量（或最终域）的修饰符，常量的值不能被修改，`static` 修饰的域的值也不能再被赋值。

- 成员方法的修饰符（常用）

1. `final`：声明方法的最终方法，即不允许子类进行覆盖。
2. `abstract`：声明方法为抽象方法，抽象方法只有方法名，没有方法体。
3. `native`：修饰为本地的方法。
4. `static`：修饰当前方法为静态方法。

### 2.3.4 静态方法（变量）和非静态方法（变量）的区别

使用 `static` 修饰的成员方法（变量）称为静态的方法（变量），这种类型的方法（变量）可在类中使用方法名（变量名）直接进行调用，而非静态的方法（变量）在使用时，必须通过 `new` 一个对象的方式，使用对象名.方法（变量名）进行调用。在类加载时，静态方法（变量）会首先加载，而动态方法（变量）只有在创建对象时才会加载。

注意：局部变量不能使用 `static` 进行修饰，因为在方法调用结束后，局部变量应该被回收，但是 `static` 影响了回收，从而保持一直存在，造成矛盾，不能使用。

静态变量在内存中只有一个内存空间，在加载类的过程中完成静态变量的内存分配，可以直接通过类名来访问。每创建一个新的实例对象，就会为实例变量分配不同的内存，各个对象访问自己的实例变量，无论创建了一个类的多少个对象，静态变量只初始化一次，所有的实例都可以访问此静态变量，而且可以通过类名直接访问

## 2.4 继承与多态

### 2.4.1 继承的概念

在很多时候，我们可以将一组相同类的属性和共有的方法封装在一个父类里面，如果一个普通类想要拥有这个类的属性和方法，则就需要使用继承。所谓继承，就是子类继承父类的属性与方法的过程。

✚ 注意：子类具有父类的一般特性（包括属性和行为），以及自身特殊的特性

✚ 继承的思想是减少代码重复率。

### 2.4.2 继承的语法

子类 `extends` 父类名

程序举例：

```
public class Student extends Person {
    public Teacher(String myName,int age) {
        super(myName, age);
    }
}
```

➤ 子类自动继承父类的属性和方法，子类中不再存在重复代码

```

public class Teacher extends Person {
    private String education; // 学历
    private String position; // 职位
    public Teacher(String myName, int age,String educ,String position) {
        super(myName, mySchool);
        this.education=educ,
        this.position=position;
    }
    public void introduction(){
        super.introduction();
        System.out.println("我的学历是： “
            + education+ “。职位是： “+position);
    }
}

```

- 老师的介绍和父类 person 不一样，可以再添加自己的特殊代码。
- 使用继承，可以有效实现代码复用

### 2.4.3 多态的概念

多态性是面向对象的特性之一，为了提高程序开发效率，降低工作量，java 实现了两种多态机制，**方法重载（静态多态）**和**方法覆盖/重写（动态多态）**。

Java 中为了使用同一个方法名，对其进行参数变化来获得一个新方法的过程称之为**方法重载**。在继承父类成员方法时，对继承过来的成员方法进行改造，这个过程就是方法覆盖（重写）。

多态的特性：重写（父类引用指向子类对象），多态的两种表现形式:方法重载和方法覆盖（方法重写）。

#### 多态中的自动转型：

数值：向上转型（自动），向下转型（不自动）

#### 为什么要转型：

当所有的子类存放在一个集合中，并且此时每一个子类都重写了其继承的一个方法，此

时，我们可以使用父类类型调用被重写的方法。

此刻如果需要该类中特有的方法，我们就需要用到向下转型来使用该类特有的成员方法。

### 向上转型（自动）程序举例：

定义一个父类，里面包含成员方法：

```
public class Person {
    public void say(){
    }
}
```

定义两个子类，并重写 say 方法，创建自己独有的方法。

```
public class Student extends Person {
    public Student() {
        super();
    }
    public void say(){}{
        System.out.println("我是一个学生");
    }
    public void program(){
        System.out.println("开发一个程序");
    }
}
```

学生类

```
public class Teacher extends Person {
    public void say(){
        System.out.println("我是一个老师");
    }
    public void skill(){
        System.out.println("展示我的才艺");
    }
}
```

老师类

```
public class Test {
    public static void main(String[] args) {
        init();
    }
}
/**
 * 初始化教师对象
 */
public static void init(){
    Person person[]=new Person[]{new Teacher(),new Teacher(),new Student(),new Student()};
    for(int i=0;i<person.Length;i++){
        person[i].say();
    }
}
}
```

结果:

```
我是一个学生
我是一个学生
我是一个老师
我是一个老师
```

结果表明, 父类指向引用子类对象时, 只要子类重写某一个方法, 我们就不用每次通过 new 对象, 并用对象来实现该方法。

**向下转型 (不自动) 举例: 在父类引用指向子类对象后, 如何使用其特有的属性。**

```
public static void init(){
    Person person[]=new Person[]{new Teacher(),new Teacher(),new Student(),new Student()};
    for(int i=0;i<person.Length;i++){
        if(person[i] instanceof Teacher){
            Teacher teacher=(Teacher)person[i];
            teacher.skill();
        }else if(person[i] instanceof Student){
            Student stu=(Student)person[i];
            stu.program();
        }
    }
}
```

结果:

```
展示我的才艺
展示我的才艺
开发一个程序
开发一个程序
```

程序结果表明, 我们父类引用指向子类对象时, 想要使用子类特有属性, 我们必须**向下转型**, 也就是**强制转型**, 才能够调用子类独有的方法。

## 2.4.4 方法重载和方法覆盖 (重写)

### ➤ 方法重载的概念

在一个类体, 当两个方法类名相同, 参数不同时, 这就构成了方法重载, 我们可以根据不同的参数类型来进行处理或者调用。

**注意:**

- 方法重载只与参数类型（数量）相关，与返回值无关。
- 如果类名相同，参数类型相同，但是参数名不同，则无法构成构造方法。

示例：

1. `public void printName(String name),`
2. `public int pringName(String s)`
3. `public void printName(String name,int id)`

1 和 3（2 和 3）可称为构造，1 和 2 不能称为构造

### ➤ 方法覆盖（重写）的概念

在类与类之间含有继承关系的时候，子类继承其父类的非私有方法，此时如果子类想要改变一下继承过来的方法的算法来适应自己的需求，这就是方法重写的概念。

程序举例：

```
public class Person {  
    public void print(){  
        System.out.println("我是父类的print方法");  
    }  
}
```

一个 person 类（父类）

```
public class Son extends Person {  
}
```

一个 Son 类继承父类，没有重写，调用 print 方法



The screenshot shows an IDE interface with a console window. The console output is: `<terminated> Test (4) [Java Application] C:\Program Files\Java\jd` followed by the Chinese text `我是父类的print方法` in a bold, black font.

结果：调用了继承过来的 print 方法

```
public class Son extends Person {  
    public void print(){  
        System.out.println("我是子类的print方法");  
    }  
}
```

一个 Son 类继承父类，重写了 print 方法



The screenshot shows an IDE interface with a console window. The console output is: `<terminated> Test (4) [Java Application] C:\Program Files\Java\jd` followed by the Chinese text `我是子类的print方法`.

结果：调用了重写后的 print 方法

## 2.4.5 关键字 this 和关键字 super

在学习和使用继承的时候，如果子类想要使用父类中的成员变量以及成员方法时，我们就可以通过 `super` 关键字来访问；如果在进行封装构造方法时，传入的参数与该类的成员变量同名时，我们则可以使用关键字 `this` 来区分参数和成员变量，构造方法之间互相调用时，我们可通过 `this (参数) /this()` 进行调用。

**注意：**

- ◇ `super ( )` 和 `this ( )` 都必须在构造方法中的第一行使用，如果不放在第一行，程序将会出现编译错误。
- ◇ 当 `super ( )` 和 `this ( )` 同时出现时，`super ( )` 在 `this ( )` 上面。
- ◇ `this` 所指的是当前类的对象，`super` 是指当前类的父类对象。

**程序举例 (this)**

```

7 public class Student {
8 private int id;//编号
9 private String name;//姓名
0 private String sex;//性别
1 private int age;//年龄
2 public Student() {
3 }
4 public Student(int id, String name, String sex, int age) {
5     this();
6     this.id = id;
7     this.name = name;
8     this.sex = sex;
9     this.age = age;
0 }

```

参数名与成员变量名相同时使用 this 来区分，构造方法之间使用 this()互相调用

## 程序举例 (super)

定义一个 person 类作为父类，包含构造方法以及公有的成员变量和成员方法

```

public class Person {
int id=15;
public Person(){
    System.out.println("我是父类的构造方法");
}
public void say(){
    System.out.println(id);
}
}

```

定义一个子类，包含构造方法（构造方法内使用 super 调用父类的构造方法），在成员方法中使用 super 调用父类的公有成员变量。

```

public class Man extends Person {
public Man(){
    super();
}
public void print(){
    System.out.println(super.id);
}
public static void main(String[] args) {
    Man man=new Man();
    man.print();
}
}

```

程序结果：

```

我是父类的构造方法
15

```

在上述例子中，我们使用 super 可以使用父类的成员变量和成员方法，这就是关键字 super 的功能。

## 2.4.6 构造方法的继承和重载

如果子类没有构造方法，则系统会为该类自动创建一个无参的构造方法，并且会在无参数的构造方法中创建一个无参数的 super，将会先执行父类无参的构造方法，后执行自己的构造方法。

如果子类拥有构造方法，且构造方法中没有调用父类构造方法的 super()语句，系统会默认创建一个无参数的 super()，那么在创建子类对象时，将会先执行父类无参的构造方法，后执行自己的构造方法。

如果子类拥有构造方法，且构造方法中有调用父类构造方法的 super 语句，那么在创建子类对象时，将会先执行父类中符合 super 参数匹配的构造方法，后执行自己的构造方法。

### 程序举例

定义一个父类 person

```
public class Person {
    public Person(){
        System.out.println("我是父类的构造方法");
    }
}
```

定义一个无构造方法的子类并生成对象

```
public class Man extends Person {
    public static void main(String[] args) {
        Man man=new Man();
    }
}
```

结果：自动生成无参构造并自动调用父类的构造方法

*我是父类的构造方法*

定义一个有构造函数但不添加 super 关键字

```

public class Man extends Person {
    public Man(){
        System.out.println("我是子类的构造方法");
    }
    public static void main(String[] args) {
        Man man=new Man();
    }
}

```

结果：总动创建 super 关键字，率先调用父类无参构造，然后调用自身构造方法

*我是父类的构造方法*  
*我是子类的构造方法*

为 person 类增加一个有参数的构造方法

```

public class Person {
    public Person(){
        System.out.println("我是父类的构造方法");
    }
    public Person(String name){
        System.out.println("我是父类包含"+name+"的构造方法");
    }
}

```

子类使用 super（参数）调用父类有参构造方法。

```

public class Man extends Person {
    public Man(){
        super("Lisan");
        System.out.println("我是子类的构造方法");
    }
    public static void main(String[] args) {
        Man man=new Man();
    }
}

```

结果：率先调用了父类包含参数的构造方法，然后调用自身构造方法

*我是父类包含Lisan的构造方法*  
*我是子类的构造方法*

- 由程序可以看出，我们可以根据不同的需求来重载构造方法，构造方法是为了创建多个入口来进行子类的成员变量和成员方法的使用。

程序举例：

```
public class Person {
    private String name;
    private int id;

    public Person() {
        super();
    }
    public Person(String name, int id) {
        super();
        this.name = name;
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
}
```

该程序使用了构造方法的重载, 我们可以通过两种创建对象的方式为该类中的私有属性进行赋值, 这就是构造方法重载的意义, 构造方法之间使用 `this` 关键字互相调用。

## 2.5 抽象类与抽象方法

### 2.5.1 抽象类的概念

使用 `abstract` 修饰符修饰的类称为抽象类, 所谓抽象类, 就是对类的抽象, 用来再被继承时, 创建一个符合该类属性的新类, 继承抽象类的子类, 必须实现抽象类中的抽象方法。

(抽象方法的概念看下一个知识点)

抽象类存在的意义就是规范一种新类的生成时的基本结构, 继承抽象类的类体必须实现抽象类中声明的抽象方法。

**抽象类举例:**

## 第三章、JAVA 常用工具类

### 3.1 Object 类

类 Object 是层次结构的根类，也就是说所有类的超类，如果一个类没有显示的指明其父类，则它的父类就是 Object 类。

➤ 常用方法

`public String toString()`，返回该对象的字符串表示。

`public int hashCode()`方法，返回该对象的哈希码值。

`public boolean equals(Object obj)`，指定某个对象是否与此对象相等。

`public final Class<?> getClass()`方法，获取当前执行的类名。

`protected Object clone()` 方法，创建并返回此对象的一个副本。

`protected void finalize()`方法，当垃圾回收机制确定不存在该对象的更多引用时，由对象的垃圾回收器调用此方法。

➤ equals 方法介绍（指定某个对象是否与此对象相等）

==与 equals()的差别：

1. ==是判断两个变量或实例是不是指向同一个内存空间，equals 是判断两个变量或实例所指向的内存空间的值是不是相同
2. ==是指对内存地址进行比较，equals()是对字符串的内容进行比较
3. ==指引用是否相同，equals () 指的是值是否相同

例子：

```
String a= "lisi" ;  
  
if(a.equals( "lisi" )){
```

```
System.out.print( "相等" );  
  
}
```

## 3.2 基本数据对象包装类

- Boolean 类, (boolean) 布尔值对象包装类。
- Character 类, char 类型数据包装类。
- Double 类, double 类型数据包装类。
- Float 类, float 类型数据包装类。
- Integer 类, int 类型数据包装类。
- Long 类, long 类型数据包装类。

所谓基本数据对象包装类, 其实是每一个基本数据类型都属于一个类, 基本的数据类型有: int,float,double,long,char, 包装类与这些数据类型都密切相关, **数据类型对象包装类**主要应用于数据转换。

作为 int 类型的包装类 Integer, 最典型的就是在 int 类型和 String 类型之间互相转换, 此外还提供处理 int 类型时常用的一些常量和方法 (下一页):

1. 常量 MIN\_VALUE 和常量 MAX\_VALUE。

```
public static final int MIN_VALUE; 其取值为 0x80000000 或  $-2^{31}$ 
```

```
public static final int MAX_VALUE; 其取值为 0x7fffffff 或  $2^{31} - 1$ 
```

2. 构造方法

```
public Integer(int value)构造一个新分配的 Integer 对象, 它表示指定的 int 值。
```

```
public Integer(String s)构造一个新分配的 Integer 对象, 它表示由 String 参数所指示的 int 值, 即由 String 对象产生一个 Integer 对象, 其中 s 代表要转换为 Integer 的 String。
```

### 3. 主要方法 (基数可理解为进制参数)

#### 1) `public static String toString(int i,int radix);`

`i` 为要转换为字符串的整数, `radix` 表示用于字符表示形式的基数, 其中第二个参数指定的基数返回第一个参数的字符串表示形式。

#### 2) `public static int parseInt(String s,int radix);`

使用第二个参数作为基数, 将字符串解析为有符号的整数。

例如:

`parseInt ("0 ", 10)`, 返回十进制数, 结果为 0;

`parseInt ("-FF ", 16)`, 返回十六进制数, 结果为-255;

#### 3) `public static Integer valueOf(String s,int radix);`

返回第二个参数提供的基数进行分析时从指定 String 中提取的值。

#### 4) `public int intValue ();`

以 `int` 类型返回该 Integer 的值, 其它的还有, `longValue ()`, `shortValue ()`, `floatValue`

`()` 以及 `doubleValue` 等等。

#### 5) `public boolean equals(Object obj);` (常用)

比较此对象与指定对象是否是同一个对象。

#### 6) `public int parseInt(String s);` (常用)

把一个数字字符转换为 `int`, 如果不是数字字符, 将会抛出异常。

`int a=Integer.parseInt( "123 ");`

类似还有 `Double.parseDouble(String s);`//转为 `int` 类型, `Float.parseFloat(String s)`

等等。

## 3.3 Java 的字符串

在编程的过程中我们往往会用到字符串这种数据结构，在 Java 中 String 类型称之为引用数据类型，Java 中将字符串分为两类：

1. 一类是创建之后不再做修改和变动的字符串常量，通常放在 String 类的对象中。
2. 一类是创建之后允许再修改和变化的字符串变量，通常放在 StringBuffer 类的对象中。

### 3.3.1 String 类

- 字符串数据类型。属于引用数据类型，String 类的特殊性：字符串为常量，字符串值一旦初始化便不可以修改。对于双引号引起来的字符串常量，系统会自动给它创建一个无名的 String 类对象。
- 常量池：在 java 用于保存在编译期已确定的，已编译的 class 文件中的一份数据。字符串特殊性内存解释。
- 常用方法：(红色为常用)

S.N.	方法 & 描述
1	<a href="#"><u>char charAt(int index)</u></a> 此方法返回指定索引处的 char 值。
2	<a href="#"><u>int codePointAt(int index)</u></a> 此方法返回指定索引处的字符（Unicode 代码点）。
3	<a href="#"><u>int codePointBefore(int index)</u></a> 此方法返回指定索引之前的字符（Unicode 代码点）。
4	<a href="#"><u>int codePointCount(int beginIndex, int endIndex)</u></a> 此方法返回在此字符串指定的文本范围的 Unicode 代码点。
5	<a href="#"><u>int compareTo(String anotherString)</u></a> 这种方法比较两个字符串字典。

6	<a href="#"><u>int compareToIgnoreCase(String str)</u></a> 这种方法比较两个字符串，字典，忽略大小写差异。
7	<a href="#"><u>String concat(String str)</u></a> 这种方法连接指定的字符串，该字符串的结束。
8	<a href="#"><u>boolean contains(CharSequence s)</u></a> 这的方法 <b>returns</b> 真正的当且仅当此字符串包含指定的 <b>char</b> 值序列。
9	<a href="#"><u>boolean contentEquals(CharSequence cs)</u></a> 这种方法比较字符串指定的 <b>CharSequence</b> 。
10	<a href="#"><u>boolean contentEquals(StringBuffer sb)</u></a> 这种方法比较字符串到指定的 <b>StringBuffer</b> 。
11	<a href="#"><u>static String copyValueOf(char[] data)</u></a> 此方法返回一个字符串，它表示的字符序列中指定数组。
12	<a href="#"><u>static String copyValueOf(char[] data, int offset, int count)</u></a> 此方法返回一个字符串，它表示的字符序列中指定数组。
13	<a href="#"><u>boolean endsWith(String suffix)</u></a> 此方法测试此字符串是否以指定后缀结束。
14	<a href="#"><u>boolean equals(Object anObject)</u></a> 这种方法比较字符串到指定的对象。
15	<a href="#"><u>boolean equalsIgnoreCase(String anotherString)</u></a> 这种方法比较字符串到另一个字符串，忽略大小写的情况考虑因素。
16	<a href="#"><u>static String format(Locale l, String format, Object... args)</u></a> 这种方法使用指定的语言环境，格式字符串和参数返回一个格式化字符串。
17	<a href="#"><u>static String format(String format, Object... args)</u></a> 这种方法使用指定的格式字符串和参数返回一个格式化字符串。
18	<a href="#"><u>byte[] getBytes()</u></a> 这个方法编码字符串到一个使用平台的默认字符集的字节序列，并将结果存储到一个新的字节数组。
19	<a href="#"><u>byte[] getBytes(Charset charset)</u></a> 这个方法编码字符串转换成一个使用给定字符集的字节序列，并将结果存储到一个新的字节数组。
20	<a href="#"><u>byte[] getBytes(String charsetName)</u></a> 这个方法编码字符串转换成使用指定的字符集的字节序列，并将结果存储到一个新

	的字节数组.
21	<a href="#">void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)</a> 这种方法从这个字符串中的字符复制到目标字符数组.
22	<a href="#">int hashCode()</a> 此方法返回该字符串的哈希码.
23	<a href="#">int indexOf(int ch)</a> 此方法返回在此字符串中第一次出现的指定字符索引.
24	<a href="#">int indexOf(int ch, int fromIndex)</a> 此方法返回索引指定的字符在此字符串中第一次出现, 在指定的索引开始搜索.
25	<a href="#">int indexOf(String str)</a> 此方法返回在此字符串中第一次出现的指定子指数.
26	<a href="#">int indexOf(String str, int fromIndex)</a> 此方法返回指数在此字符串中第一次出现的指定子, 开始在指定的索引.
27	<a href="#">String intern()</a> 该方法返回一个字符串对象的规范表示.
28	<a href="#">boolean isEmpty()</a> 此方法返回 <code>true</code> , 当且仅当 <code>length()</code> 为 <code>0</code> .
29	<a href="#">int lastIndexOf(int ch)</a> 此方法返回在此字符串中最后一次出现的指定字符索引.
30	<a href="#">int lastIndexOf(int ch, int fromIndex)</a> 此方法返回的索引, 搜索指定的字符在此字符串中最后一次出现在指定的索引开始向后.
31	<a href="#">int lastIndexOf(String str)</a> 此方法返回的最右边出现的指定子字符串在此指数.
32	<a href="#">int lastIndexOf(String str, int fromIndex)</a> 此方法返回在此字符串中最后一次出现的指定子索引, 搜索在指定的索引开始向后.
33	<a href="#">int length()</a> 此方法返回此字符串的长度.
34	<a href="#">boolean matches(String regex)</a> 这个方法告诉这个字符串是否匹配给定的正则表达式.
35	<a href="#">int offsetByCodePoints(int index, int codePointOffset)</a>

	此方法返回在此字符串中的偏离给定的索引 <code>codePointOffset</code> 代码点指数.
36	<a href="#">boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)</a> 这种方法的测试，两个字符串区域是否相等的情况下忽略.
37	<a href="#">boolean regionMatches(int toffset, String other, int ooffset, int len)</a> 该方法测试两个 <code>string</code> 区域是否是相等.
38	<a href="#">String replace(char oldChar, char newChar)</a> 此方法返回一个新的字符串替换此字符串中出现的所有 <code>oldChar</code> 与 <code>newChar</code> .
39	<a href="#">String replace(CharSequence target, CharSequence replacement)</a> 此方法替换每个子字符串，该字符串指定的文字替换序列相匹配的文字靶序列.
40	<a href="#">String replaceAll(String regex, String replacement)</a> 该方法与给定的替换此字符串匹配给定的正则表达式替换每个子.
41	<a href="#">String replaceFirst(String regex, String replacement)</a> 该方法与给定的替换，替换第一个给定的正则表达式的字符串相匹配的子串.
42	<a href="#">String[] split(String regex)</a> 该方法拆分给定的正则表达式匹配的串在一起围.
43	<a href="#">String[] split(String regex, int limit)</a> 该方法拆分给定的正则表达式匹配的串在一起围.
44	<a href="#">boolean startsWith(String prefix)</a> 该方法的测试，如果这个字符串用指定的前缀开始.
45	<a href="#">boolean startsWith(String prefix, int toffset)</a> 该方法的测试，如果在指定的索引开始的子字符串，该字符串开始用指定的前缀.
46	<a href="#">CharSequence subSequence(int beginIndex, int endIndex)</a> 此方法返回一个新的字符序列，这个序列的一个子.
47	<a href="#">String substring(int beginIndex)</a> 该方法返回一个新的字符串，这个字符串的子串.
48	<a href="#">String substring(int beginIndex, int endIndex)</a> 该方法返回一个新的字符串，这个字符串的子串.
49	<a href="#">char[] toCharArray()</a> 此方法转换字符串到一个新的字符数组.
50	<a href="#">String toLowerCase()</a>

	此方法转换所有的字符在此字符串中使用的默认语言环境的规则，以小写的情况下。
51	<a href="#">String toLowerCase(Locale locale)</a> 此方法转换所有使用给定的 <b>Locale</b> 的规则，在此字符串中的字符小写。
52	<a href="#">String toString()</a> 此方法返回的字符串本身。
53	<a href="#">String toUpperCase()</a> 该方法将在此字符串中的所有字符为大写使用的默认语言环境的规则。
54	<a href="#">String toUpperCase(Locale locale)</a> 该方法将在此字符串中的所有字符为大写的规则给定的 <b>Locale</b> 。
55	<a href="#">String trim()</a> 该方法返回一个字符串的副本，开头和结尾的空格省略。
56	<a href="#">static String valueOf(boolean b)</a> 此方法返回的布尔型参数的字符串表示形式。
57	<a href="#">static String valueOf(char c)</a> 此方法返回的 <b>char</b> 参数的字符串表示形式。
58	<a href="#">static String valueOf(char[] data)</a> 此方法返回 <b>char</b> 数组参数的字符串表示形式。
59	<a href="#">static String valueOf(char[] data, int offset, int count)</a> 此方法返回一个特定的子数组的 <b>char</b> 数组参数的字符串表示形式。
60	<a href="#">static String valueOf(double d)</a> 此方法返回的字符串表示形式的 <b>double</b> 参数。
61	<a href="#">static String valueOf(float f)</a> 此方法返回 <b>float</b> 参数的字符串表示形式。
62	<a href="#">static String valueOf(int i)</a> 此方法返回 <b>int</b> 参数的字符串表示形式。
63	<a href="#">static String valueOf(long l)</a> 此方法返回 <b>long</b> 参数的字符串表示形式。
64	<a href="#">static String valueOf(Object obj)</a> 此方法返回一个对象参数的字符串表示形式。

### 3.3.2 StringBuffer

➤ 概念:

字符串缓冲区, 缓冲区用于存储数据, 所以也称之为容器。字符串的组成原理就是通过该类实现的。

➤ StringBuffer 与 String 的区别:

效率更快, 避免了过多字符串常量垃圾对象的产生,前者用于生成字符串, 后者用于表示字符串,前者为常量, 后者是“可变化的量”。

➤ 主要方法

构造方法摘要

1. StringBuffer()

构造一个其中不带字符的字符串缓冲区, 其初始容量为 16 个字符。

2. StringBuffer(CharSequence seq)

`public java.lang.StringBuilder(CharSequence seq)` 构造一个字符串缓冲区, 它包含与指定的 `CharSequence` 相同的字符。

3. StringBuffer(int capacity)

构造一个不带字符, 但具有指定初始容量的字符串缓冲区。

4. StringBuffer(String str)

构造一个字符串缓冲区, 并将其内容初始化为指定的字符串内容。

主要方法:

1. `public String toString()`

返回此序列中数据的字符串表示形式。

2. `public StringBuffer append(boolean b)` 及重载

将指定类型参数的字符串表示形式追加到序列。

3. `public StringBuffer insert(int offset, boolean b)`及其重载方法

将指定类型参数的字符串表示形式插入此序列中。

4. `public StringBuffer delete(int start, int end)`

移除此序列的子字符串中的字符。

5. `public StringBuffer deleteCharAt(int index)`

返回此序列中指定索引处的 `char` 值。

6. `public StringBuffer replace(int start, int end, String str)`

使用给定 `String` 中的字符替换此序列的子字符串中的字符。

7. `public StringBuffer reverse()`

8. `public String substring(int start)`

9. `public String substring(int start, int end)`

### 3.3.4 StringBuilder

字符串缓冲区

方法与 `StringBuffer` 完全相同,效率更快,线程不安全

## 3.4 Math 类

- `Math` 类的概念

`Math` 类是 `java` 提供给我们的数学运算的类, 里面提供了一些常用的数学运算方法,

大致可分为 4 个类别: 分别为三角函数方法、指数函数方法取整函数方法以及最大值、最

小值和绝对值函数方法。

### 3.4.1 三角函数方法

static double	<a href="#">acos</a> (double a) 返回一个值的反余弦；返回的角度范围在 0.0 到 $\pi$ 之间。
static double	<a href="#">asin</a> (double a) 返回一个值的反正弦；返回的角度范围在 $-\pi/2$ 到 $\pi/2$ 之间。
static double	<a href="#">atan</a> (double a) 返回一个值的反正切；返回的角度范围在 $-\pi/2$ 到 $\pi/2$ 之间。
static double	<a href="#">atan2</a> (double y, double x) 将矩形坐标 (x, y) 转换成极坐标 (r, <i>theta</i> )，返回所得角 <i>theta</i> 。
static double	<a href="#">cbrt</a> (double a) 返回 double 值的立方根。
static double	<a href="#">ceil</a> (double a) 返回最小的（最接近负无穷大）double 值，该值大于等于参数，并等于某个整数。
static double	<a href="#">copySign</a> (double magnitude, double sign) 返回带有第二个浮点参数符号的第一个浮点参数。
static float	<a href="#">copySign</a> (float magnitude, float sign) 返回带有第二个浮点参数符号的第一个浮点参数。
static double	<a href="#">cos</a> (double a) 返回角的三角余弦。
static double	<a href="#">cosh</a> (double x) 返回 double 值的双曲线余弦
static double	<a href="#">toDegrees</a> (double angrad) 将用弧度表示的角转换为近似相等的用角度表示的角。
static double	<a href="#">toRadians</a> (double angdeg) 将用角度表示的角转换为近似相等的用弧度表示的

### 3.4.2 指数函数方法

static double	<a href="#">exp</a> (double a) 返回欧拉数 $e$ 的 double 次幂的值。
static double	<a href="#">log</a> (double a) 返回 double 值的自然对数（底数是 $e$ ）。
static	<a href="#">log10</a> (double a)

double	返回 double 值的底数为 10 的对数。
static double	<a href="#">sqrt(double a)</a> 返回正确舍入的 double 值的正平方根
static double	<a href="#">cbrt(double a)</a> 返回 double 值的立方根。
static double	<a href="#">pow(double a, double b)</a> 返回第一个参数的第二个参数次幂的值。

### 3.4.3 取整函数方法

static double	<a href="#">ceil(double a)</a> 返回最小的（最接近负无穷大）double 值，该值大于等于参数，并等于某个整数。
static double	<a href="#">floor(double a)</a> 返回最大的（最接近正无穷大）double 值，该值小于等于参数，并等于某个整数。
static double	<a href="#">rint(double a)</a> 返回最接近参数并等于某一整数的 double 值。
static int	<a href="#">round(float a)</a> 返回最接近参数的 int。

### 3.4.4 取最大值、最小值、绝对值函数

static int	<a href="#">min(int a, int b)</a> 返回两个 int 值中较小的一个。
static int	<a href="#">max(int a, int b)</a> 返回两个 int 值中较大的一个。
static int	<a href="#">abs(int a)</a> 返回 int 值的绝对值。

在以上的数据运算函数中，数据类型可变，double、float、int 都可使用

### 3.4.5 Math 中定义的常量

static double	<a href="#">E</a> 比任何其他值都更接近 $e$ （即自然对数的底数）的 double 值。
static double	<a href="#">PI</a> 比任何其他值都更接近 $\pi$ （即圆的周长与直径之比）的 double 值。

### 3.4.6 Math.random() 函数

在我们进行程序编写的过程中,经常会用到随机数来设置随机数来保证游戏或者一些公平性的问题,java 中给我们提供了随机数的产生函数来简单我们的程序开发流程。

Math.Random()方法主要返回一个 0.0~1.0 之间的随机数,返回带正号的 double 值,该值大于等于 0.0 且小于 1.0。

使用方法

```
double num=Math.random();//获取一个 0~1 之间的随机小数
```

```
int num=Math.floor(Math.random()*10)//获取一个 0~10 (不包含 10) 之间的随机数。下取整为整数。
```

### 3.4.7 大数据运算

为了防止大数据类型的精度丢失和数据空间不足的问题,java 中提供给我们一个大数据运算的类,即 BigInteger 和 BigDecimal,其中 BigInteger 主要是处理大整型数据,而 BigDecimal 主要针对大的小数类型的数据处理。

➤ BigInteger 类的两个构造函数

1. public BigInteger(String str);//str 必须是十进制字符串
2. BigInteger twoInstance=new BigInteger( "2" );//将十进制 2 转换为 BigInteger
3. BigInteger(byte[] val) 将包含 BigInteger 的二进制补码表示形式的 byte 数组转换为 BigInteger。

➤ BigInteger 类的主要方法 (常用)

<a href="#">BigInteger</a>	<b>abs()</b> 返回其值为此 BigInteger 的绝对值的 BigInteger。
----------------------------	---

<a href="#">BigInteger</a>	<b><a href="#">add(BigInteger val)</a></b> 返回其值为 (this + val) 的 BigInteger。
int	<b><a href="#">compareTo(BigInteger val)</a></b> 将此 BigInteger 与指定的 BigInteger 进行比较。
double	<b><a href="#">doubleValue()</a></b> 将此 BigInteger 转换为 double。
boolean	<b><a href="#">equals(Object x)</a></b> 比较此 BigInteger 与指定的 Object 的相等性。
float	<b><a href="#">floatValue()</a></b> 将此 BigInteger 转换为 float。
int	<b><a href="#">hashCode()</a></b> 返回此 BigInteger 的哈希码。
int	<b><a href="#">intValue()</a></b> 将此 BigInteger 转换为 int。
long	<b><a href="#">longValue()</a></b> 将此 BigInteger 转换为 long。
<a href="#">BigInteger</a>	<b><a href="#">max(BigInteger val)</a></b> 返回此 BigInteger 和 val 的最大值。
<a href="#">BigInteger</a>	<b><a href="#">min(BigInteger val)</a></b> 返回此 BigInteger 和 val 的最小值。
byte[]	<b><a href="#">toByteArray()</a></b> 返回一个 byte 数组，该数组包含此 BigInteger 的二进制补码表示形式。
<a href="#">String</a>	<b><a href="#">toString()</a></b> 返回此 BigInteger 的十进制字符串表示形式。
<a href="#">String</a>	<b><a href="#">toString(int radix)</a></b> 返回此 BigInteger 的给定基数的字符串表示形式。

<code>static</code>	<b><code>valueOf(long val)</code></b> 返回其值等于指定 long 的值的 <code>BigInteger</code> 。
<code>BigInteger</code>	<b><code>subtract(BigInteger val)</code></b> 返回其值为 <code>(this - val)</code> 的 <code>BigInteger</code> 。
<code>BigInteger</code>	<b><code>multiply(BigInteger val)</code></b> 返回其值为 <code>(this * val)</code> 的 <code>BigInteger</code> 。
<code>BigInteger</code>	<b><code>divide(BigInteger val)</code></b> 返回其值为 <code>(this / val)</code> 的 <code>BigInteger</code> 。
<code>BigInteger</code>	<b><code>remainder(BigInteger val)</code></b> 返回其值为 <code>(this % val)</code> 的 <code>BigInteger</code> 。

➤ `BigDecimal` 的常用构造函数

<b><code>BigDecimal(double val)</code></b> 将 <code>double</code> 转换为 <code>BigDecimal</code> ，后者是 <code>double</code> 的二进制浮点值准确的十进制表示形式。
<b><code>BigDecimal(String val)</code></b> 将 <code>BigDecimal</code> 的字符串表示形式转换为 <code>BigDecimal</code> 。

➤ 常用的加减乘除方法

<code>BigDecimal</code>	<b><code>add(BigDecimal augend)</code></b> 返回一个 <code>BigDecimal</code> ，其值为 <code>(this + augend)</code> ，其标度为 <code>max(this.scale(), augend.scale())</code> 。
<code>BigDecimal</code>	<b><code>subtract(BigDecimal subtrahend)</code></b> 返回一个 <code>BigDecimal</code> ，其值为 <code>(this - subtrahend)</code> ，其标度为 <code>max(this.scale(), subtrahend.scale())</code> 。
<code>BigDecimal</code>	<b><code>multiply(BigDecimal multiplicand)</code></b> 返回一个 <code>BigDecimal</code> ，其值为 <code>(this × multiplicand)</code> ，其标度为 <code>(this.scale() + multiplicand.scale())</code> 。
<code>BigDecimal</code>	<b><code>divide(BigDecimal divisor)</code></b> 返回一个 <code>BigDecimal</code> ，其值为 <code>(this / divisor)</code> ，其首选标度为 <code>(this.scale() - divisor.scale())</code> ；如果无法表示准确的商值（因为它有无穷的十进制扩展），则抛出 <code>ArithmeticException</code> 。

## 3.5 Date 类

Java 中提供给用户对日期类型数据的产生和操作的类, Date 类属于 java.util 包下, 在 JDK1.0 中, Date 类是唯一的一个代表时间的类, 但是由于 Date 类不便于实现国际化, 所以从 JDK1.1 版本开始, 推荐使用 Calendar 类进行时间和日期处理。这里简单介绍一下 Date 类的使用。

### 3.5.1 构造方法

**Date date=new Date();//唯一的未过时的构造方法**

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

使用 Date 类的默认构造方法创建出的对象就代表当前时间, 由于 Date 类覆盖了 toString 方法, 所以可以直接输出 Date 类型的对象, 显示的结果如下:

```
Sun Mar 08 16:35:58 CST 2009
```

在该格式中, Sun 代表 Sunday(周日), Mar 代表 March(三月), 08 代表 8 号, CST 代表 China Standard Time(中国标准时间, 也就是北京时间(东八区))。

#### 2、使用 Date 类代表指定的时间

```
Date d1 = new Date(2009-1900,3-1,9);
```

```
System.out.println(d1);
```

使用带参数的构造方法, 可以构造指定日期的 Date 类对象, Date 类中年份的参数应该是实际需要代表的年份减去 1900, 实际需要代表的月份减去 1 以后的值。例如上面的示例代码代表就是 2009 年 3 月 9 号。

实际代表具体的年月日时分秒的日期对象, 和这个类似。

### 3.5.2 常用的方法（未过时）

boolean	<a href="#">after(Date when)</a> 测试此日期是否在指定日期之后。
boolean	<a href="#">before(Date when)</a> 测试此日期是否在指定日期之前。
<a href="#">Object</a>	<a href="#">clone()</a> 返回此对象的副本。
int	<a href="#">compareTo(Date anotherDate)</a> 比较两个日期的顺序。
boolean	<a href="#">equals(Object obj)</a> 比较两个日期的相等性。
int	<a href="#">hashCode()</a> 返回此对象的哈希码值。
void	<a href="#">setTime(long time)</a> 设置此 Date 对象，以表示 1970 年 1 月 1 日 00:00:00 GMT 以后 time 毫秒的时间点。
<a href="#">String</a>	<a href="#">toString()</a> 把此 Date 对象转换为以下形式的 String： dow mon dd hh:mm:ss zzz yyyy 其中： dow 是一周中的某一天 (Sun, Mon, Tue, Wed, Thu, Fri, Sat)。

### 3.5.3 日期类型的其他类

#### 存储时间常用类：

util.Date:父类，精确到秒

sql.Date：子类，精确到日，Date.valueOf("2016-3-3");

Timestamp：子类，精确到秒以后，Timestamp.valueOf("2016-3-3 12:2:3");

**操作时间数据常用类：**

Calendar：提供了一些关于日期的算法，可以实现对日期内容的增删改查；

setTime(Date date) date--Calendar,getTime()

➤ **Sql.Date 构造方法**

Date(long date) 使用给定毫秒时间值构造一个 Date 对象。

➤ **Sql.Date 常用方法**

void	<a href="#">setTime</a> (long date) 使用给定毫秒时间值设置现有 Date 对象。
<a href="#">String</a>	<a href="#">toString</a> () 格式化日期转义形式 yyyy-mm-dd 的日期。
static <a href="#">Date</a>	<a href="#">valueOf</a> (String s) 将 JDBC 日期转义形式的字符串转换成 Date 值。

➤ **Timestamp 类的构造方法**

## 第四章、JavaWeb 基础

Java Web 应用由一组静态 HTML 页、Servlet、JSP 和其他相关的 class 组成。每种组件在 Web 应用中都有固定的存放目录。

### 4.1 Web 运行环境 Tomcat 配置

#### 4.1.1 名词解释

Tomcat 服务器是一个免费的开放源代码的 Web 应用服务器,属于轻量级应用服务器,在中小型系统和并发访问用户不是很多的场合下被普遍使用,是开发和调试 JSP 程序的首选。由 Apache、Sun 和其他一些公司及个人共同开发而成。由于有了 Sun 的参与和支持,最新的 Servlet 和 JSP 规范总是能在 Tomcat 中得到体现, Tomcat 5 支持最新的 Servlet 2.4 和 JSP 2.0 规范。因为 Tomcat 技术先进、性能稳定,而且免费,因而深受 Java 爱好者的喜爱并得到了部分软件开发商的认可,成为目前比较流行的 Web 应用服务器。

#### 4.1.2 目录结构及用途

我们使用的是 Tomcat 绿色免安装版的,我们只需要将压缩文件进行解压,我们进行解压之后,会看到如下的几个目录。

目录	用途
bin	包含启动/关闭脚本
conf	包含不同的配置文件,包括 server.xml(Tomcat 的主要配置文件)和为不同的 Tomcat 配置的 web 应用设置缺省值的文件 web.xml
doc	包含各种 Tomcat 文档
lib	包含 Tomcat 使用的 jar 文件.unix 平台此目录下的任何文件都被加到 Tomcat 的

	classpath 中
Logs	存放 Tomcat 的日志文件
/server	包含 3 个子目录: classes、lib 和 webapps
src	用户创建的 java 文件存放位置
webapp	包含 web 项目示例, 当发布 web 应用时, 默认情况下把 web 文件夹放于此目录下
work	Tomcat 自动生成, 放置 Tomcat 运行时的临时文件(如编译后的 JSP 文件). 如在 Tomcat 运行时删除此目录. JSP 页面将不能运行. [jsp 生成的 sevlet 放在此目录下]
classes	你可以创建此目录来添加一些附加的类到类路径中. 任何你加到此目录中的类都可在 Tomcat 的类路径中找到自身.
Common/bin	存在 Tomcat 服务器及所有的 web 应用程序可以访问的 JAR 文件
Server/bin	存在 Tomcat 服务器运行所需的各种 JAR 文件。
Share/Bin	存在所有的 web 应用程序可以访问的 JAR 文件(不能被 tomcat 访问)
/server/webapps	存放 tomcat 两个自带 Web 应用 admin 应用和 manager 应用

### 4.1.3 JAR 存放位置

Server/bin, Share/Bin, Common/bin 目录下都可以放 JAR, 他们的区别在于:

在 Server/bin 目录下的 JAR 文件只能被 Tomcat 服务器访问。在 Share/Bin, 目录下的 JAR 文件可以被所有的 web 应用程序访问, 但不能被 Tomcat 服务器访问。在 Common/bin 目录下的 JAR 文件可以被 Tomcat 服务器和所有的 web 应用程序访问。

此外, Java Web 应用程序中, 在它的 WEB-INF 目录下, 也可以建立 lib 子目录, 在 lib 子目录下可以存放各种 JAR 文件, 这些 JAR 文件只能被当前 WEB 应用程序所访问。Web 应用的配置信息存放在 web.xml 文件中。在发布某些组件(如 Servlet)时, 必须在 web.xml 文件中添加相应的配置信息。

在 Tomcat 应用服务器上发布 Web 应用程序, 应该在 <CATALINA\_HOME>/webapps 目录下创建这个 Web 应用的目录结构。

目 录	描 述
/helloapp	Web 应用的根目录，所有的 JSP 和 HTML 文件都存放于此目录下
/helloapp/WEB-INF	存放 Web 应用的发布描述文件 web.xml
/helloapp/WEB-INF/classes	存放各种 class 文件，Servlet 类文件也放于此目录下
/helloapp/WEB-INF/lib	存放 Web 应用所需的各种 JAR 文件。例如，在这个目录下，可以存放 JDBC 驱动程序的 JAR 文件。

注：在 classes 以及 lib 子目录下，都可以存放 Java 类文件。在运行过程中，Tomcat 的类装载器先装载 classes 目录下的类，再装载 lib。

#### 4.1.4 Tomcat 的配置文件

Tomcat 的配置基于两个配置文件：

1. server.xml - Tomcat 的全局配置文件
2. web.xml - 在 Tomcat 中配置不同的关系环境

##### server.xml

server.xml 是 Tomcat 的主配置文件.完成两个目标:

- 1 提供 Tomcat 组件的初始配置.
- 2 说明 Tomcat 的结构,含义,使得 Tomcat 通过实例化组件完成启动及构建自身

##### web.xml

Tomcat 可以让用户通过将缺省的 web.xml 放入 conf 目录中来定义所有关系环境的 web.xml 的缺省值.建立一个新的关系环境时,Tomcat 使用缺省的 web.xml 文件作为基本设置和应用项目特定的 web.xml(放在应用项目的 WEB-INF/web.xml 文件)来覆盖这些缺省值.

### 4.1.5 环境搭建

我们在配置 tomcat 时，需要配置环境变量，使得 tomcat 能够正常使用，需要配置的环境变量如下：

1. JAVA\_HOME: JDK 的安装路径
2. CATALINA\_HOME: Tomcat 的安装目录

#### ➤ 启动闪屏的解决办法

编辑 startup.bat 第二行 添加

Jdk 的安装目录: set JAVA\_HOME=C:\Program Files\Java\jdk1.8.0\_45

Tomcat 的安装目录: set CATALINA\_HOME=E:\apache-tomcat-7.0.61

·编辑 shutdown.bat 第二行 添加 set

JAVA\_HOME=C:\ProgramFiles\Java\jdk1.8.0\_45

startup.sh,shuwdown.sh 是在 linux 下启动的

#### ➤ 端口号地址被占用

Address already in use: JVM\_Bind

cmd -->查看本机所有应用端口情况 netstat -ano,查找指定应用的端口号的进程 ID:

netstat -ano|findstr "端口号",taskkill 是用来终止进程的 /F 指定要强行终止的进程。 /IM 立即终止进程号

强行终止进程: taskkill /F /IM PID

#### ➤ 测试环境

打开解压缩后的 Tomcat 文件夹，双击打开 startup.bat，等待出现启动的毫秒数，在浏览器地址栏键入如下：<http://localhost:8080> 或者 [http:// 本机 IP:8080](http://本机IP:8080)，如果出现 Tomcat 的欢迎界面，则配置正常。

## 4.2 JSP 常用内置对象

JSP 全名为 Java Server Pages, 中文名叫 java 服务器页面, 其根本是一个简化的 Servlet 设计, 它是由 Sun Microsystems 公司倡导、许多公司参与一起建立的一种动态网页技术标准。JSP 技术有点类似 ASP 技术, 它是在传统的网页 HTML (标准通用标记语言的子集) 文件(\*.htm,\*.html)中插入 Java 程序段(Scriptlet)和 JSP 标记(tag), 从而形成 JSP 文件, 后缀名为(\*.jsp)。用 JSP 开发的 Web 应用是跨平台的, 既能在 Linux 下运行, 也能在其他操作系统上运行。