变量

1.1. 什么是变量

在日常生活中，人们会用到大量数据，像去ATM机取款，首先，需要插入银行卡，这个过程其实就是ATM机的系统在获取银行卡号这个数 据，而后，输入密码，这个过程也是在获取数据，也就是密码。在后续进行的业务处理中，像存钱、取钱、转帐汇款等等，银行卡号和密码会被反复的、频繁的使 用， 那就需要一个存储这些数据的地方。

在软件系统中，是将数据存储在内存之中的，而对内存中的数据的引用就是变量，可以理解为变量就是内存中数据的代词。

简单说， 变量就是指代在内存中开辟的存储空间，用于存放运算过程中需要用到的数据。代码如下所示：

int a = 5;

int b = 6; //6;字面量

int c = a + b;

如上代码中，变量a、b、c指代内存中三块用于存储整数的存储空间，分别用于存储两个整数及这两个整数之和，int意为整数数据类型，后续课程会详细介绍。

对于变量，我们需要关注如下几个方面：

1.变量的声明：用特定语法声明一个变量，让运行环境为其分配空间。

2.变量的命名：需要有个见名知意的名字，而且要符合Java语言规范。

3.变量的初始化：变量声明后，要为其赋一个确定的初值后再使用。

4.变量的访问：可以对变量中的数据进行存取、操作，但必须和其类型匹配。

1.2. 变量的声明

1.2.1. 变量的声明

当需要使用一个变量时，必须对该变量进行声明，变量的声明包含两点：变量名和数据类型，代码如下所示：

int a

上面的代码中，int为变量的数据类型，a为变量的名称，当声明如上语句时，JVM会为该变量在内存中开辟存储空间，不同的变量类型决定了存储空间的结构（后面详细介绍）。

1.2.2. 未经声明的变量不能使用

Java语言语法规定，变量使用之前必须声明，否则会有编译错误。代码如下所示：

public static void main(String[] args) {

 a = 1;

// 编译错误，变量没有声明

int score = 0；

scord = 100；

// 编译错误

System.out.println(score);

}

从上面的代码中，可以看到出现了两个编译错误，编译错误即为javac过程出现的错误，主要是由于语法问题导致的。第一个错误，a =1，是因为该变量没有声明。第二个错误，scord=100，是因为前面声明的为score，编译器并未找到scord变量，该错误是因为拼写错误造成的。

1.2.3. 一条语句中声明多个同类型变量

如果多个变量的类型一样，可以在一条语句中声明，中间使用逗号分隔， 代码如下所示：

public static void main(String[] args) {

int a=1, b=2;

int c, d=3;

}

从上面的代码中，可以看到，第一条语句，声明了两个整型变量，分别赋值为1和2，中间使用逗号分隔，最后以；号结尾。第二条语句，声明了两个整型变量，c没有赋初始值，d赋初值为3。

1.3. 变量的命名

1.3.1. 命名需要符合标识符语法要求

在java语言中，对于***变量、常量、方法、类、包***等等都有名字，将这些名字统一称之为**java标识符，**标识符的命名规则如下列表所示：

1. 可以由字母、数字、“\_”或“$”符组成，但是不能以数字开头。**字母宽泛概念** 见名知意 类首字母大写 一般第一个小写第二个单词大写
2. 中文可以作为变量名，但不提倡使用。
3. Java大小写敏感，即：严格区分大小写，在给命名变量时需要注意。
4. 不能使用Java保留字（一些Java语言规定好的，有特殊含义的字符），如：int、if、else、for、break等。

看如下的一些实例，可以体现出java标识符的命名规则：

int 123go = 100； // 编译错误，标识符不能以数字开头。

int 成绩 = 60； // 编译没错，标识可以是中文，但不建议使用。

int break= 200； // 编译错误，break是Java保留字。

int score = 80;

System.out.println(Score);// 编译错误，Java大小写敏感，Score变量没有声明。

1.3.2. 命名需见名知意，且符合Java规范

Java 变量的命名应“见名知意”，同时，Java编程规范要求：变量的命名需采用“驼峰命名法”，即如果变量的名字有多个单词组成，除***第一个单词外，其他单词的 首字母大写，其余的字母小写***，例如：salary、 javaScore 、studentName、empSalary 等。

1.4. 变量的初始化

1.4.1. 未经初始化的变量不能使用

Java语法规定：变量在使用之前必须初始化，即必须给该变量赋予特定的值，而在C语言中，变量使用之前可以不需要初始化，但是，其初始值不确定，而Java语言的设计者为了避免因此而带来的错误，规定变量必须初始化之后才能使用，代码如下所示：

public static void main(String[] args) {

int a, b = 10;

int c = a + b; // 编译错误

System.out.prinltn(c);

}

上面的代码出现了编译错误， 原因是变量b赋了初始值，在给变量c赋值时可以使用b，而变量a并未赋初始值就直接使用了，违反了java语法的规定，变量使用之前必须初始化，所以出现编译错误。

1.4.2. 在声明变量时初始化

可以在变量声明时初始化，语法：变量类型 变量名称 = 初始值；代码如下所示：

public static void main(String[] args) {

int sum = 0; //声明同时初始化

int a = 5;

int b = 6;

sum = a + b;

System.out.println(sum);

}

1.4.3. 在第一次使用变量前初始化

可以在变量声明以后，通过赋值语句对变量进行初始化，但一定确保在第一次使用该变量之前，代码如下所示：

public static void main(String[] args) {

int sum;

sum = 0; // 在使用sum变量之前对其进行初始化。

sum = sum + 100;

System.out.println(sum);

}

1.5. 变量的访问

1.5.1. 可以对变量中的值进行存取，操作

变量，顾名思义，可以改变的量，在java语言中，可以对其赋值、更改等操作，但是需要注意的是，对变量的操作即为对其所存储的数据的操作。代码如下所示：

public static void main(String[] args) {

int a = 100;

a = a + 200; //该条语句的义：将变量a中的值加上200所得结果再存入变量a

}

1.5.2. 变量的操作必须与类型匹配

变量在声明时指定了它的数据类型， Java编译器会检测对该变量的操作是否与其类型匹配，如果对变量的赋值或者操作与其类型不匹配，会产生编译错误。代码如下所示：

public static void main(String[] args) {

int salary;

salary = 15000.50; // 编译错误，整型变量不可以赋予浮点值（小数）。

double d = 123.456;

int n = d%2; // 编译错误，d%2结果为double型，不能赋给整型的n。

}

1Kb=1024B

1B=8bit（位）

Java中基本类型所占的字节与平台无关

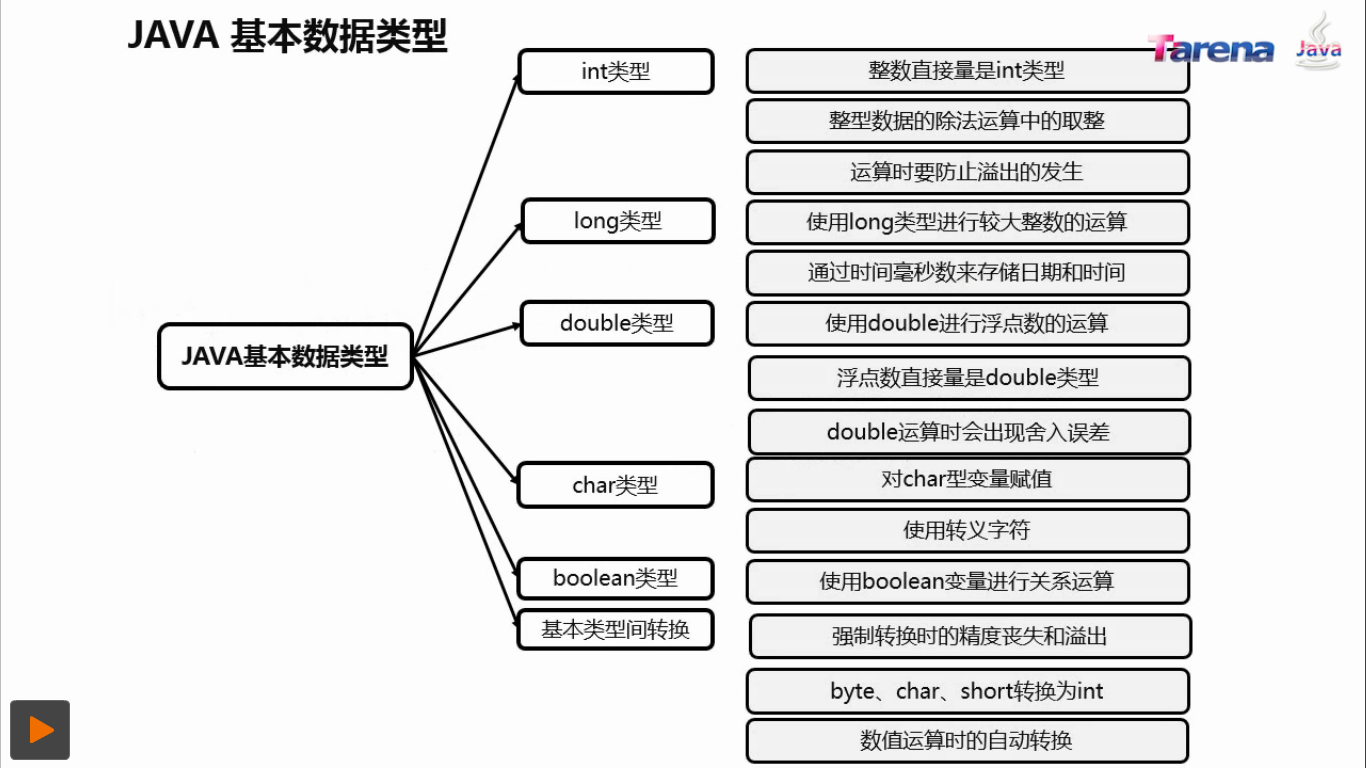
Java数据类型：

基本数据类型

引用数据类型（类类型）;

**2. JAVA基本类型(四类八种)**

**2.1. 8种基本数据类型**



Java语言有8种基本数据类型， 分别用于**存储整数**、**浮点数**、**字符数据**和**布尔类型数据**。需要注意的是： 现在所介绍的仅仅是基本数据类型，后续还会介绍很多非基本数据类型。基本数据类型如图 – 1所示：

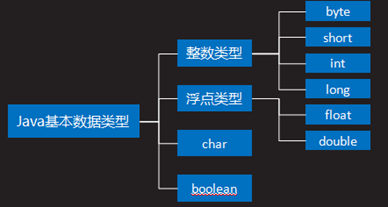


图- 1

从图- 1中可以看出， 基本数据类型主要分为4大类 （**整数类型、浮点类型、char、boolean**）， 整数类型又分为了4小类（**byte、short、int、long）**, 浮点类型也分了2小类**（float、double）**，这些数据类型的区别是怎样的？通过图– 2展示了这8种数据类型的存储空间及使用场景：



图- 2

在如上这8种数据类型中，最常用的有5种，分别为int、long、double、char、boolean。其余的数据类型几乎不用，要求对这5种基本数据类型重点掌握，其余的数据类型，有兴趣了解就可以了。

**基本数据类型（四种）**

Byte(字节)

1字节（8位） 2^(8-1)

第一个是符号位 剩余7位0111 1111（8421）

-2^7 ­­­—2^7-1 =》 -128——127

Short(短整形)

2字节（16位） 2^(16-1)

第一位是符号位 剩余15位 0111 1111 1111 1111

-2^15—2^15 -1 =》 （-32768）——（32767）

int(整形)

4字节（32位） 2^(32-1)

第一位是符号位 剩余31位 0111 1111 1111 1111…一共31个1 一个0

-2^31—2^31 -1 =》 （- 2147483648）­­——（2147483647）

Long(长整形)

2字节（64位） 2^(64-1)

第一位是符号位 剩余63位 0111 1111 1111 1111… 一共63个1 一个0

-2^63—2^63 -1

long num4=123456777L; //L是long的缩写

**浮点数据类型**

遵循IEEE754协议标准

浮点数不能表示一个精确小数

浮点数在计算机计算时特别费资源

除法难乘法简单

float(单精度) ------------比long大很少使用

输出7位小数，有限位

数据的符号位

占1位 =>1bit

指数部分

占8位

一位控制正负 =》1bit

七位控制数大小 (2^7)-1 2^127 =》8bit

尾数部分 =》23bit

float fnum1=3.141592654653464643f;

//但精度在结尾必须加f/F 大小写都可以；7位

Double(双精度) 8字节 64位

double dnum1=3.141592654653464643d;

//双精度在结尾可以加d/D 大小写都可以也可以不加；

**字符类型**

Char 2字节  **16bit 整数 非负数 16位全部用来表示整数**

**0 - 65535**

一个字符串

//字符类型 0-65535 unicode

**布尔类型**

Boolean 1字节（1位真或假）1bit false/true;

字符串类型 字符串

**2.2. int类型**

**2.2.1. int类型**

int是最常用的整数类型，一个int类型的变量占用4个字节，即32位的内存空间。Int的最大表示范围为：-231～231-1，即-2147483648 ~2147483647，大约正负21个亿多些。

**2.2.2. 整数直接量是int类型**

所谓整数直接量（literal）就是直接写出的整数，例如：下面的语句中，100就是直接量。

1. int a = 100；

关于整数直接量，需要注意如下要点：

1. **整数的直接量的类型默认为int类型**，如果直接写出的整数超过了int的表达范围，将会出现编译错误，下面的语句，就是因为超出了整数的范围而导致的编译错误。
2. int d = 10000000000; // 编译错误10000000000这个数值写出来就是错误的，因为Java认为所有直接写出的整数都是int类型，而这个数值超过了int的表达范围。
3. 除了通常的十进制数字形式，整数直接量也可以写成16进制的形式（以0X或0x开头）或8进制的形式（以0开头），请看如下直接量三种表现形式：
4. int a = 100000; // 10进制
5. int b = 0x186a0; // 16进制
6. int c = 0303240; // 8进制

**2.2.3. 整型数据除法运算中的取整**

若对两个整数相除，会舍弃小数的部分（注意：不是四舍五入），结果也是整数。示例代码如下所示：

1. int c = 5/3;
2. System.out.println(c); // c的值为1，取整
3. int total = 87;
4. int error = 23;
5. int percent = error / total \* 100;
6. System.out.println(percent+"%"); //结果为0%，23除以87整数部分为0，乘以100，为0
7. percent = 100 \* error / total;
8. System.out.println(percent + "%"); // 结果为26%，230除以87整数部分为26

**2.2.4. 运算时要防止溢出的发生**

当两个整数进行运算时， 其结果可能会超过整数的范围而发生溢出，正数过大而产生的溢出，结果为负数；负数过大而产生的溢出，结果为正数。示例代码如下所示：

1. int a = 2147483647; //int类型整数的上限
2. int b = -2147483648; //int类型整数的下限
3. a = a + 1;
4. b = b - 1;
5. System.out.println("a=" + a); //输出结果： a=-2147483648 溢出，结果错误。
6. System.out.println("b=" + b); //输出结果： b=2147483647溢出，结果错误。

**2.3. long类型**

**2.3.1. long类型**

在表示整数时，如果int类型的范围不够，可以使用long类型，一个long型的变量占用8个字节（即64位），最大表示范围为：-2^63 ~ 2^63-1，即 -9223372036854775808 ~ 9223372036854775807。

如果要表示long直接量，需要以 L 或 l 结尾。示例代码如下：

1. long a = 10000000000; //会有编译错误，因为10000000000编译器认为是int类型，而这个值，已经超出了int的范围
2. long c=2147483648L;
3. long b = 10000000000l; //正确

**2.3.2. 使用long类型进行较大整数的运算**

对于较大的整数运算（超过int的表达范围），可以使用long型。示例代码如下：

1. long distance1 = 10000 \* 365 \* 24 \* 60 \* 60 \* 299792458l;
2. //必须有一个long型数据参与的运算结果才是long型
3. System.out.println("distance1="+distance1);//distance1=547836957965889536 结果正确
4. long distance2 = 10000 \* 365 \* 24 \* 60 \* 60 \* 299792458;
5. System.out.println("distance2="+ distance2); //distance2=-1973211136 溢出，=号后面的数据默认为int类型，超出了范围，发生溢出。

**2.3.3. 通过时间毫秒数来存储日期和时间**

JDK提供 System.currentTimeMillis() 方法，返回1970年1月1日零点**到此时此刻所经历的毫秒数**，数据太大，故其数据类型为long。示例代码如下：

Long手册

1. long time = System.currentTimeMillis();
2. System.out.println(time); //输出的结果为： 1383835712828

通过上面的代码可以看出，输出的结果已经超出int类型的最大值，因此，JDK设计的返回类型为long型，该方法常常被用于计时操作。

**2.4. double类型**

**2.4.1. 使用double进行浮点数的运算**

前面所学习的int、long都是用于存储整数的，小数即为浮点数，包括： float（单精度）和double（双精度），double类型的精度值是float类型的两倍，因此而得名双精精，在实际的应用开发中，float应 用极少，大多数场合使用double表示浮点数。示例代码如下：

1. double pi = 3.14;
2. double r = 8;
3. double s = pi \* r \* r;
4. System.out.println("s=" + s); // 输出的结果为：s=200.96

**2.4.2. 浮点数直接量是double类型**

浮点数的直接量有两种写法：1）通常写法，如：3.14、314、0.1、.5。 2）科学计数法，如：1.25E2、1.25e2、1.25E-2。其中，1.25E2表示1.25乘以10的2次方。

默认的浮点直接量为double型，如果需要表示float类型的直接量，需要加“f”或“F”后缀。例如：

1. **float f1 = 3.14 //编译错误，应该写成3.14f**

**正确方法 float f2=3.14f;**

**2.4.3. double运算时会出现舍入误差**

2进制系统中无法精确的表示1/10，就好像十进制系统中无法精确的表示1/3一样，

所以，2进制表示10进制会有一些舍入误差，对于一些要求精确运算的场合会导致代码的缺陷。示例代码如下所示：

1. double money = 3.0;
2. double price = 2.9;
3. System.out.println(money - price); //输出的结果是： 0.10000000000000009

如果需要精确的运算可以考虑放弃使用double或float而采用BigDecimal 类来实现。关于这一点，将在后续的章节中介绍。

**2.5. char类型**

**2.5.1. char类型**

**字符类型char事实上是一个16位无符号整数（都是正数），**这个值是对应字符的编码，Java字符类型采用**Unicode字符集编码 （通用码、统一码、万国码）**，而Unicode是世界通用的定长字符集，**所有的字符都是16位来表示**。例如：字符a实际的值为97，字符A实际的值为 65，字符0实际的值为48。

字符直接量可以采用诸如：‘中’的形式，也可以采用16进制的表示形式，例如： ‘\u4e2d’，代码如下所示：

1. char c1 = ‘中’; //c1中存的是”中”的编码
2. char c2 = '\u4e2d'; //‘4e2d’为‘中’所对应的16位Unicode编码的16进制表示形式
3. System.out.println(c1);
4. System.out.println(c2);

如上代码的输出结果：c1的值为中，c2值也为中，但c1和c2内部存储的其实是”中”这个字符所对应的Unicode码，即：一个无符号的整数。

**2.5.2. 对char型变量赋值**

在对char型变量赋值时，可以采用如下三种方式：

方式一：

字符直接量：形如‘A’，变量中实际存储的是该字符的Unicode编码（无符号整数值），一个char型变量只能存储一个字符。示例如下：

1. char c1 = 'A';

方式二：

整型直接量：范围在0~65535之间的整数，变量中实际存储的即该整数值，但表示的是该整数值所对应的Unicode字符。示例如下：

1. char c2 = 65;

Unicode形式：形如‘\u0041’,Unicode字符的16进制形式。示例如下：

1. char c3 = '\u0041';

**2.5.3. 使用转义字符**

字符直接量需要包含在一对’’单引号之中，那如果想表示单引号’的字符时，需要怎么表示？想表示回车、换行符时，怎么表示？

因为单引号为特殊意义的字符， 那么，对于不方便输出的字符可以采用转义字符来表示，示例代码如下：

1. char c = '\\';
2. System.out.println(c); //输出的结果为：\

常用转义字符如下图 – 2所示：

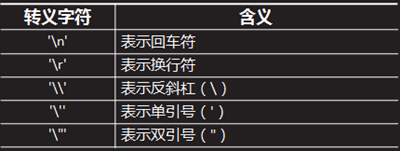


图- 2

**2.6. boolean类型**

**2.6.1. 使用boolean变量进行关系运算**

boolean类型适用于关系、逻辑运算， 表示某个条件是否成立， 只允许取值true或false，true表示条件成立， 而false表示条件不成立。

boolean型变量经常用于存储关系运算的结果，所谓关系运算就是比较两个变量的大小相等等关系（此知识点，后续详细介绍）。boolean示例代码如下所示：

1. int age = 18;
2. boolean isChild = age<16;
3. System.out.println(isChild); // isChild的值为false
4. boolean running = true;
5. boolean closed = false;

**2.7. 基本类型间转换**

**2.7.1. 类型间转换**

不同的基本类型直接可以相互转化，主要有两种方式：

1. 自动类型转化（**隐式类型转换）：**从小类型到大类型可以自动完成。类型的大小关系如下图 - 3所示：

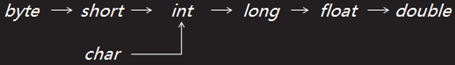


图- 3

1. **强制转化：从大类型到小类型需要强制转换符**，语法如下： 反向转换
2. （需要转换成的类型）变量

因为大类型的精度值大于小类型，取值范围大于小类型，所以，当使用强制转化时，有可能会造成精度的损失或者溢出，所以，在使用强制转化时要求显式的告诉编译器，正在进行强制转换。

**2.7.2. 强制转换时的精度丧失和溢出**

基本类型转化如下示例所示，注意强制转换时可能会造成的精度丧失和溢出。

1. int a = 100;
2. int b = 200;
3. long c = a + b; //自动将int转化为long
4. long l1 = 1024l;
5. int i = (int) l1; //需要加强制转化符由于1024在int的范围内，所以没有产生溢出
6. long l = 1024L \* 1024 \* 1024 \* 4;
7. int j = (int) l; //会产生溢出
8. System.out.println(j); // 结果为：0
9. double pi = 3.1415926535897932384;
10. float f = (float) pi;  **7位**  //会造成精度的损失，因为单精度的精确度小于double
11. System.out.println(f); //结果为：3.1415927

**2.7.3. 数值运算时的自动转换**

**如果在一个表达式中出现了多种数据类型，则运算结果会自动的向较大的类型进行转化**，

示例如下：

1. //由于有long型的直接量参与，整个表达式的结果为long
2. long distance = 10000 \* 365 \* 24 \* 60 \* 60 \* 299792458L;
3. **//由于有double型的直接量599.0参与，整个表达式的结果为 double**
4. **double change = 800 - 599.0;**
6. **//结果为0.0，右边都是int型数据运算结果也为int类型，结果为0，再赋值给double**
7. 型，将0转化为 0.0
8. double persent1 = 80 / 100;
10. //结果为0.8，右边表达式有double型直接量参与， 运算结果为double型
11. double persent2 = 80.0 / 100;

**2.7.4. byte、char、short转换为int**

在前面所介绍的8种数据类型中，byte、char、short、int、long都表示整数类型，而整型的直接量为int，在实际使用中，为了方便使用，遵循了如下的规则：

1. **int直接量可以直接赋值给byte、char和short，只要不超过其表示范围。**示例如下：
2. byte b = 97;
3. short s = 97;
4. char c = 97;
5. **btye b4=15+16; //31**
6. byte、char、short三种类型参与运算时，先一律转换成int类型再进行运算。示例如下：
7. byte b = 97;
8. int num = b + b; //num的值为194

**3. 运算符和表达式-1**

**3.1. 算数运算**

**3.1.1. 使用%运算符**

Java算术运算符除了通常的加（+）、减（-）、乘（\*）、除（\）之外，还包括取模运算（%）和自增（++）及自减（--）运算。+，-，\*，/ 比较简单，实验一下即可。

取模运算（%）意为取余数，只能适用于整数及char类型。示例代码如下所示：

1. // 输出255除以8所得的余数。
2. int n = 225;
3. System.out.println(n % 8); //结果为1

**3.1.2. 使用“++”和“--”运算符**

Java的自增运算符（++）和自减运算符（--）继承自C++，可以使变量的值加1或减1，但其写在变量前和变量后有不同的效果：

1. 如果写在变量前，表示在使用这个变量之前加1或减1
2. 如果写在变量后，表示这个变量使用完之后再加1或减1

示例代码如下所示：

1. int a = 10, b = 20;
2. int c1 = a++; // 先将a的值赋给c1，然后a再自加
3. int c2 = ++b; // 先将b的值自加，然后再赋给c2
4. System.out.println("a=" + a + ", b=" + b + ", c1=" + c1 + ", c2=" + c2);
5. // 输出的结果为： a=11, b=21, c1=10, c2=21

**3.2. 关系运算**

**3.2.1. 使用关系运算符**

Java中的关系运算符用于判断数据之间的大小关系，包括大于（>）、小于（<）、大于等于（>=）、小于等于（<=）、等于（==）、不等于（!=） 六个运算符。

在实际应用中常常使用，假设现在想实现转帐功能，输入转帐的金额，需要判断所转帐的金额是否大于当前帐户的金额，那结果只有两个，大于 或不大于，在java语言中，使用boolean类型来表示这种是否的状态，如果关系成立为true，否则为false。示例代码如下所示：

1. int max = 10;
2. int num = 9;
3. boolean b1 = max > 15;
4. boolean b2 = num%2 == 1;
5. System.out.println(b1); // 结果为false
6. System.out.println(b2); // 结果为true

**3.3. 逻辑运算**

**3.3.1. 逻辑运算**

前面的关系运算符是用于比较两个数值之间的大小关系的， 而逻辑运算符是用来进行逻辑运算的，它是建立在关系运算的基础之上的。当两个关系运算需要兼顾考虑时，可以使用逻辑运算符。

逻辑运算符包括：与（&&）、或（||）和非（!）。参与逻辑运算的变量或表达式都是boolean类型，运算结果也为boolean类型。逻辑运算规则如下图 - 4所示：

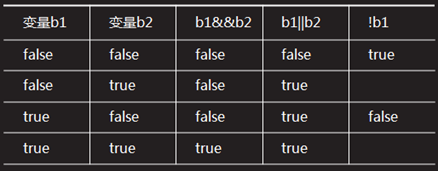


图- 4

通过分析可以看出：

1. 当变量b1和变量b2同时为true时，&&表达式为true。
2. 当变量b1和变量b2有一个为true时，||表达式为true。
3. ！只有一个表达式，当b1为true时，!b1为false，b1为false时，!b1为true。

**3.3.2. 使用“&&”运算符**

两个boolean变量参与“&&”运算时，只有当两个变量均为true的时，运算结果才为true，否则结果为false。示例代码如下所示：

1. int score = 80;
2. boolean b = score >= 60 && score < 90;
3. System.out.println(b);
4. // 结果为true，因为score的值同时满足大于等于60和小于90这两个条件，逻辑表达式“score >= 60”和“score < 90”的结果均为true，所以&&运算的结果即为true。

**3.3.3. 使用“||”运算符**

两个boolean变量参与“||”运算时，当两个变量有一个为true时，结果即为true，只有当两个变量均为false时结果为false。示例代码如下所示：

1. boolean flag = true;
2. int n = 200;
3. boolean b1 = flag || (n >= 0 && n < 100);
4. System.out.println(b1);
5. // 结果为true，上面这段代码中，表达式“flag || (n >= 0 && n < 100)”的含义是：当flag为true或者n在0到100之间（n大于等于0且小于100）时，结果为true，否则为false。根据flag和n的值，最后的运算结果为true。

**3.3.4. 使用“!”运算符**

“!”运算相对简单，只有一个boolean变量参与运算，运算的值与该变量相反，变量为true时结果为false，变量为false是结果为true。示例代码如下所示：

1. boolean flag = true;
2. int n = 200;
3. boolean b = !flag || (n >= 0 && n < 100);
4. System.out.println(b);
5. // 结果为false，上面这段代码中，表达式“!flag || (n >= 0 && n < 100)”的含义是：当flag为false或者n在0到100之间（n大于等于0且小于100）时，结果为true，否则为false。根据flag和n的值，最后的运算结果为false。

**3.3.5. 关于“短路逻辑”的问题**

Java逻辑运算中的&&和||有短路的特性，当第一个关系表达式就可以判断出整个表达式的结果时，就不会再去判断后面的第二个表达式。

1. 对于“&&”，当第一个操作数为false时，将不会判断第二个操作数，因为此时无论第二个操作数是什么最后的结果一定是false；
2. 对于“||”，当第一个操作数为true时，将不会判断第二个操作数，因为此时无论第二个操作数为何，最后的运算结果一定是true。

示例代码如下所示：

1. int i = 100, j = 200;
2. boolean b1 = (i > j) && (i++ > 100);
3. System.out.println(b1); // 结果为：false
4. System.out.println(i); // 结果为：100，发生短路，i++不会被执行
5. boolean b2 = i > 0 || j++ > 200;
6. System.out.println(b2); // 结果为：true
7. System.out.println(j); // 结果为：200，发生短路，j++不会被执行