第三章 操作符、流程控制

1 变量

> 在之前的学习，我们已经了解过变量了，这里先对变量进行一些总结

变量的作用：

1. 程序中使用变量可以接收、保存、传递、操作数据
2. 变量的类型和数据的类型必须是一致的
3. 如果类型不一致，那么就需要进行类型转换（自动转换、手动转换）

变量的使用：

1. 必须是先声明、再赋值、再使用
2. 也可以声明变量的同时就进行赋值，然后再使用
3. 如果只声明变量，没有赋值，那么再使用的时候会报错

变量的种类：

1. 通过类型划分
    - 基本类型变量（byte short int long float double char boolean）
    - 引用类型变量（类类型、接口类型、数组类型）
2. 通过范围划分
    - 局部变量
    - 实例变量

> 注意，在不同的情况给一个变量可以赋不同的值，那么这个变量就可以表示不同的数据，也就是说，随着程序的运行，变量所表示的值，是可以随时发生变化的

例如，

```
public void test(){
    int a;

    a = 1;
    System.out.println(a); //输出变量a的值为1

    a = 2;
    System.out.println(a); //输出变量a的值为2

    a = 3;
    System.out.println(a); //输出变量a的值为3

}
```

例如，

```java
public void test(int age){
    String msg;
    if(age >= 18){
        msg = "你好~年轻人";
    }
    else{
        msg = "你好~小朋友";
    }
    System.out.println(msg);
}
```

> 通过观察上面代码，你能确定变量msg最后输出的是什么值么？（体会、理解变量的本质）

1.1 局部变量

> 定义在方法中的变量，就是局部变量

例如，

```java
public void test(){
    //int类型的变量a，同时也是一个局部变量
    int a = 1;
}
```

局部变量没有默认值：

```java
public void test(){
    int a;
    //编译报错，因为局部变量a没有赋值，也没有默认值，那么就不能使用变量a的值
    System.out.println(a);
}
```

局部变量的作用范围：

1. 变量都是有作用范围的，超出这个范围，变量就不能使用了
2. 局部变量被**直接包裹**的大括号中，从这变量声明开始，一直到这个大括号结束

例如，

```java
public void test(){//大括号开始

    int a = 1;
    {
        //这里可以使用变量a
    }
    //这里可以使用变量a

}//大括号结束
```

> 注意观察，变量a是被“**直接包裹**”在哪一个大括号中？

例如，

```
public void test(int x){

    int a = 1;
    {
        int b = 2;
        {
            int c = 3;
        }

    }

}
```

> 说一说，变量a b c他们的作用范围，分别是从哪里到哪里？

注意，方法的参数，也是局部变量，它的作用范围在整个方法中

1.2 实例变量

> 实例变量就是类中的属性，也叫做成员变量（非静态的）

实例变量是默认值的，即使声明完，不赋值，它也是有一个默认的值：

```
public class Student{

    public String name;
    public int age;

    public void print(){
        System.out.println(name);//可以使用，成员变量有默认值
        System.out.println(age);//可以使用，成员变量有默认值
    }

}
```

不同类型的实例变量，它们的默认值是：

- byte类型，默认值为0
- short类型，默认值为0
- int类型，默认值为0
- long类型，默认值为0L
- float类型，默认值为0.0F
- double类型，默认值为0.0D
- boolean类型，默认值false

- char类型，默认值是'\u0000'
- 引用类型，默认值是null

例如，

```
public class VariableDemo {
    public byte     v1;
    public short    v2;
    public int      v3;
    public long     v4;
    public float    v5;
    public double   v6;
    public boolean  v7;
    public char     v8;
    public String   v9;

    public void showValue(){
        System.out.println("v1 = "+v1);
        System.out.println("v2 = "+v2);
        System.out.println("v3 = "+v3);
        System.out.println("v4 = "+v4);
        System.out.println("v5 = "+v5);
        System.out.println("v6 = "+v6);
        System.out.println("v7 = "+v7);
        System.out.println("v8 = "+v8);
        System.out.println("v9 = "+v9);
    }

    public static void main(String[] args) {
        VariableDemo demo = new VariableDemo();
        demo.showValue();
    }

}
```

实例变量的作用范围是当前类中所有的 `非静态` 方法中，都可以访问。

2 操作符

2.1 赋值操作符

操作符	作用	例子
=	最基础的赋值操作符，=号右边的值，赋给=号左边变量	int a = 1; int x = 0;
*=	一个变量和另一个数据相乘，并把结果再赋值给这个变量	int a = 1; a*=2; //a = a*2;
/=	一个变量和另一个数据相除，并把结果再赋值给这个变量	int a = 2; a/=2; //a = a/2;
%=	一个变量和另一个数据相余，并把结果再赋值给这个变量	int a = 5; a%=2; //a = a%2;
+=	一个变量和另一个数据相加，并把结果再赋值给这个变量	int a = 5; a+=2; //a = a+2;
-=	一个变量和另一个数据相减，并把结果再赋值给这个变量	int a = 5; a-=2; //a = a-2;

除此之外，还有<<=　>>=　>>>=　&=　^=　|=，和上面的含义是一样的，了解过<<　>>　>>>　&　^　|这几个二进制操作后，那么加上=号的意思也就知道了。

一些特殊的参数，a+=1 可以写成a++ ，表示a变量自增1的操作。

同样的 a-=1 可以写成a--，表示a变量自减1的操作。

a++ 和 ++a 的区别：

- a++ 表示**先**使用a的值进行操作或者运算，**然后**再让a完成自增1
  例如，

```
int a = 1;
int b = a++;
System.out.println(a);//输出 2
System.out.println(b);//输出 1
```

- ++a 表示**先**让a完成自增1，**然后**再使用a的值进行操作或者运算
  例如，

```
int a = 1;
int b = ++a;
System.out.println(a);//输出 2
System.out.println(b);//输出 2
```

a-- 和 --a的区别和上面是类似的。

2.2 比较操作符

操作符	作用	例子
>	比较是否大于	1>0
>=	比较是否大于等于	1>=0
<	比较是否小于	1<2
<=	比较是否小于等于	1<=2
instanceof	判断对象是否属于指定类型	stu instanceof Student

instanceof，判断一个指定的对象，是否【属于】另一个指定的类型

例如，

```
Person o = new Person();
//表示引用o所指向的对象，是不是属于Person类型
System.out.println( o instanceof Person );
```

2.3 相等操作符

操作符	作用	例子
==	比较两边的数据是否相等，相等返回true，不相等返回false	1==2，o1 == o2
!=	比较两边的数据是否不相等，相等返回false，不相等返回true	1!=2，o1!=o2

> 可以用在俩个数字之间的判断，也可以用俩个对象之间的判断。

2.4 算术操作符

操作符	作用	例子
+	数字之间使用+,表示俩个值相加	int a = 1+1;
-	两个数字相减	int a = 1-1;
*	两个数字相乘	int a = 1*1;
/	两个数字相除	int a = 1/1;
%	两个数字取余	int a = 5%2;

> 注意，使用+号，也可以连接（拼接）俩个字符串，数值也可以和字符串使用+号连接，连接之后的结果还是字符串

2.5 移位操作符

操作符	作用	例子
>>	算术右移位运算，也叫做【带】符号的右移运算	8 >> 1
<<	左移位运算	8 << 1
>>>	逻辑右移位运算，也叫做【不带】符号的右移运算	8 >>> 1

>> 算术右移位运算

注意，这个操作的本质就是除以2^n,这个n就是我们右移的位数。

注意，除以2^n之后，只保留整数部分
注意，正数右移之后，最左边空出的位置，都要补0
注意，负数右移之后，最左边空出的位置，都要补1

例如，16 >> 3 结果是2 ，相当于 16 / 2^3 = 2

<< 左移位运算

注意，这个操作的本质就是乘以2^n,这个n就是我们左移的位数
注意，无论正数负数左移之后，最右边空出的位置，都要补0
注意，当左移之后，得到的数字已经超出当前类型所能表示的最大值的时候，这个值最终会被限定到这个当前类型中，所以最终显示的值会和我们逻辑上算出的值有所不同。
例如：直接使用2进制表示数字

```
int a = 0b01000000000000000000000000000000;
int result = a<<2; //其实这个结果已经超出了int能表示的最大值
System.out.println(result); //结果是0
```

特殊情况：

```
int a = 0b00000000000000000000000000000001;
System.out.println(a<<32);  //结果是1 相当于1<<0
System.out.println(a<<33);  //结果是2 相当于1<<1
System.out.println(a<<34);  //结果是4 相当于1<<2
```

原因：
如果移动的位数超过了该类型的最大位数，那么编译器会对移动的位数取模/取余。如果对int型移动33位，实际上只移动了33%32=1位。如果对int型移动32位，实际上只移动了32%32=0位

>>> 逻辑右移位运算，也叫做【不带】符号的右移运算

注意，这个操作的本质就是除以2^n,这个n就是我们右移的位数

注意，除以2^n之后，只保留整数部分
注意，正数和负数右移之后，最左边空出的位置，都要补0

例如：

```
12>>>1  结果是 6
-12>>>1 结果是 2147483642
```

注意：在操作的时候，java操作的都是计算机中的补码

> 正数的原码、反码、补码都是一样的
> 例如：数字1的原码0000 0001，反码0000 0001，补码0000 0001
>
> 负数的原码、反码、补码有所不同
> 例如：数字-1
> 原码：1000 0001
> 反码：1111 1110 除了符号位之外，其他按位取反
> 补码：1111 1111 反码基础上加1

2.6 位运算符

操作符	作用	例子
&	与运算	1&1=1，1&0=0， 0&1=0， 0&0=0
|	或运算	1|1=1，1|0=1，0|1=1，0|0=0
^	异或运算	1^1=0， 0^0=0， 1^0=1， 0^1=1，相同为0，不同为1
~	取反运算	0 -> 1 ，1 -> 0

例如，交互俩个变量的值，但是不是要中间变量

```
//使用中间变量的情况
int a = 1;
int b = 2;
int temp = a; //a:1 b:2 temp:1
a = b;//a:2 b:2 temp:1
b = temp;//a:2 b:1

//不是使用中间变量，使用+ -操作
int a = 1;
int b = 2;

a = a+b;//a:3 b:2
b = a-b;//a:3 b:1
a = a-b;//a:2 b:1


//不是使用中间变量，使用^操作
int a = 1;
int b = 2;

a = a^b;
b = a^b;
a = a^b;

```

2.7 逻辑运算符

&& 短路与
例如，

```
int a = 1;
int b = 5;
boolean result;

// a>4 这布尔表达式为false
// 后的(b++)>1就不需要计算了
// 因为当前是短路与（&&）运算，第一个false已经能够决定整个表达式的结果了
result = a>4 && (b++)>1;
System.out.println(result);//false
System.out.println(b);//输出5

// 这种情况下，a>0为true
// 必须要再进行后面表达式的计算，最终才能得到结果，所以要计算后的(b++)>1的部分
result = a>0 && (b++)>1;
System.out.println(result);//true
System.out.println(b);//输出6

```

|| 短路或，和上面&&的操作类似，||的逻辑是：第一个布尔表达式为true，才能决定整个表达式的结果

`& 和 && 有什么区别？`

&既可以作为二进制数字的位运算符，也可以作为布尔表达式中的逻辑运算符，但是作为逻辑运算符的时候，&并没有&&符合的那种**短路**的功能。

&& 只能作为逻辑运算符，但是它会具备短路的功能。

注意，|和||的区别也是类似的；

2.8 条件操作符

> 也可以称为三目运算符

语法：
boolean表达式？表达式1：表达式2

例如，将x和y中，较大的数赋值给变量z

```
int x = 10;
int y = 5;
int z;
z = (x > y) ? x : y;//三目运算符
```

2.9 优先级问题

以上介绍操作符的操作，是有优先级的，具体如下：

优先级	运算符	结合性
1	()、[]、{}	从左向右
2	!、+、-、~、++、--	从右向左
3	*、/、%	从左向右
4	+、-	从左向右
5	«、»、>>>	从左向右
6	<、<=、>、>=、instanceof	从左向右
7	==、!=	从左向右
8	&	从左向右
9	^	从左向右
10	|	从左向右
11	&&	从左向右
12	||	从左向右
13	?:	从右向左
14	=、+=、-=、*=、/=、&=、|=、^=、~=、«=、»=、>>>=	从右向左

3 类型转换

> java中的=号赋值操作，需要=号俩边的类型一致，也就是=号右边的数据的类型要和=号左边的变量的类型保持一致，如果不一致，那么就需要做类型的转换，分为隐式转换和显示转换。

隐式转换也称为自动转换。

显示转换也称为强制转换/手动转换。

3.1 基本类型

隐式转换（Implicit），也是自动转换。

在JVM运行期间，只要满足条件，就可以自动完成类型转换的过程。一般是数据范围比较小的，自动就可以转换为数据范围比较大的类型（基本类型）。

例如，

```
byte a = 1;
int b = a; //注意，这里在运行期间，就自动完成了转换的过程
```

显示转换（explicit），也是手动转换/强制转换。（简称：强转，有风险）

编译器发现类型无法自动转换的情况，就会编译报错，这时候我们确认无误后，就可以进行类型强制转换。

但是这里是存在一定风险的，在基本类型数据中，这个风险主要是可能数据有损失，在引用类型中，将来在运行这个类型转换代码的时候，有可能会抛出类型转换异常。

例如，

```
int a = 100;
//编译报错，因为a是int类型，b是byte
//把32位int数据，赋值给8位的变量b
//把一个范围大的数据，赋给一个范围小的变量
//这时候是不允许的，也无法类型自动转换。
byte b = a;

//编译通过，自己手动进行了类型转换
//对于基本类型，强制转换就是把多余的位给抹去
//所以这时候可能对数据的值，造成影响
byte b = (byte)a;

```

> 注意，浮点型数据，如果强行转换为整型，小数部分就需要全部抹去。

例如，生成[0,9]直接的随机数（整数）

```
//Math.random()返回[0,1)的随机数，类型是double
double random = Math.random()*10;
int a = (int)random;
System.out.println(a);
```

3.2 引用类型

隐式转换（Implicit）

例如，

```
Student s = new Student();
Object o = s;//特点：子类类型的变量，可以自动转为（隐式）父类类型

//上面俩句代码可以合成这一句代码，其实就是把中间变量s给去掉了。
Object o = new Student();

```

显示转换（explicit）

例如，

```
Object o = new Student();
Student s = (Student)o;//特点：父类类型的变量，需要强制转为（显式）子类类型
```

思考1，一个子类型的变量，一定能自动转为它的父类型么？

思考2，一个父类型变量，一定能强制转成它的一个子类型么？

可以结合着instanceof关键字进行考虑。

4 流程控制

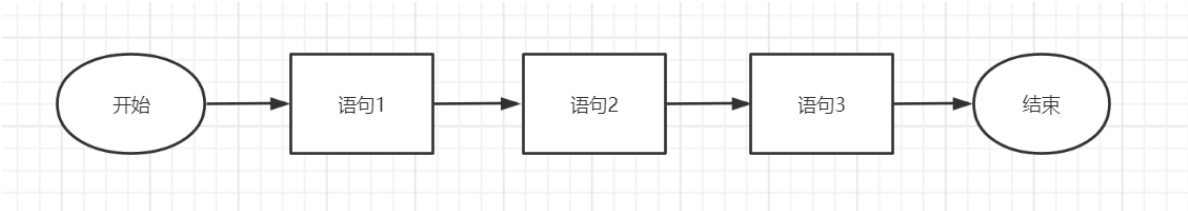
在一个程序执行的过程中，各条语句的**执行顺序**对程序的结果是有直接影响的。所以，我们必须清楚每条语句的执行流程。而且，很多时候要通过控制语句的执行顺序来实现我们想要的功能。

4.1 概述

程序中，需要执行的代码，其结构主要分为

- 顺序结构
- 分支结构
- 循环结构

其中，顺序结构就是最基本的流程控制，只要将代码从上到下，按照顺序依次编写即可，大多数代码都是这样的结构，如图：



4.1 if

语法1：

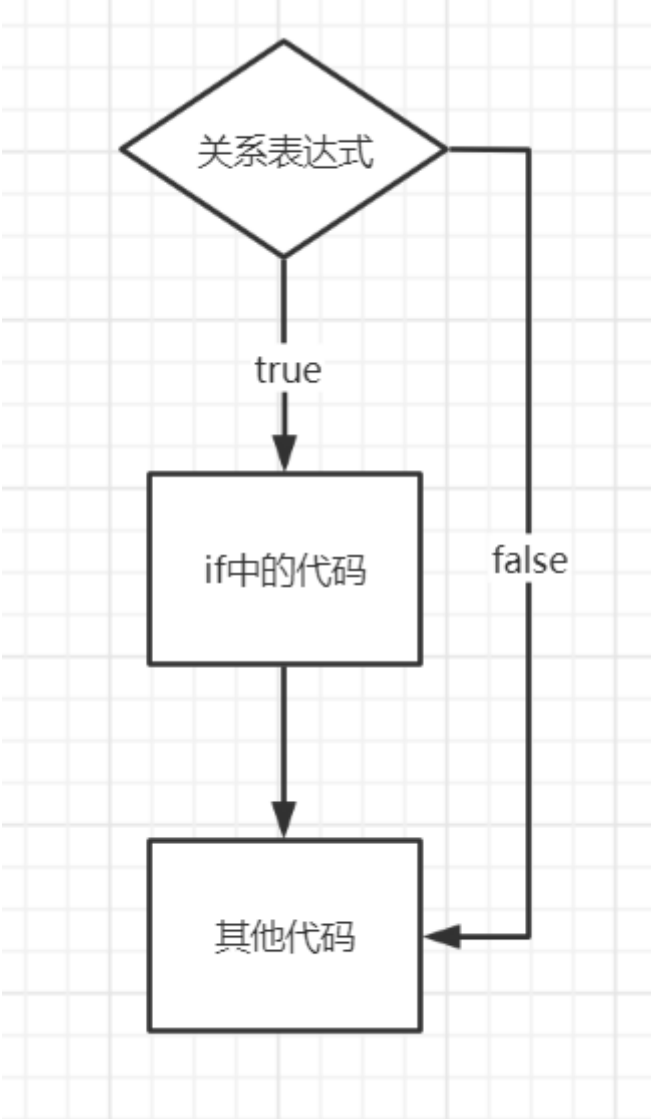
```
if (关系表达式) {
    语句体;
}
//其他代码
```

执行流程：

1. 首先计算 **关系表达式** 的值
2. 如果关系表达式的值为true，就执行语句体
3. 如果关系表达式的值为false，则不执行语句体

4. 继续执行if代码块后面的其他代码

如图：



例如，

```
int a = 10;
if(a%2==0){
    System.out.println("变量a的值为偶数");
}
//其他代码
```

> if中的代码是否执行，主要是看这里的布尔表达式的结果，如果是true则执行代码，如果是false则不执行。

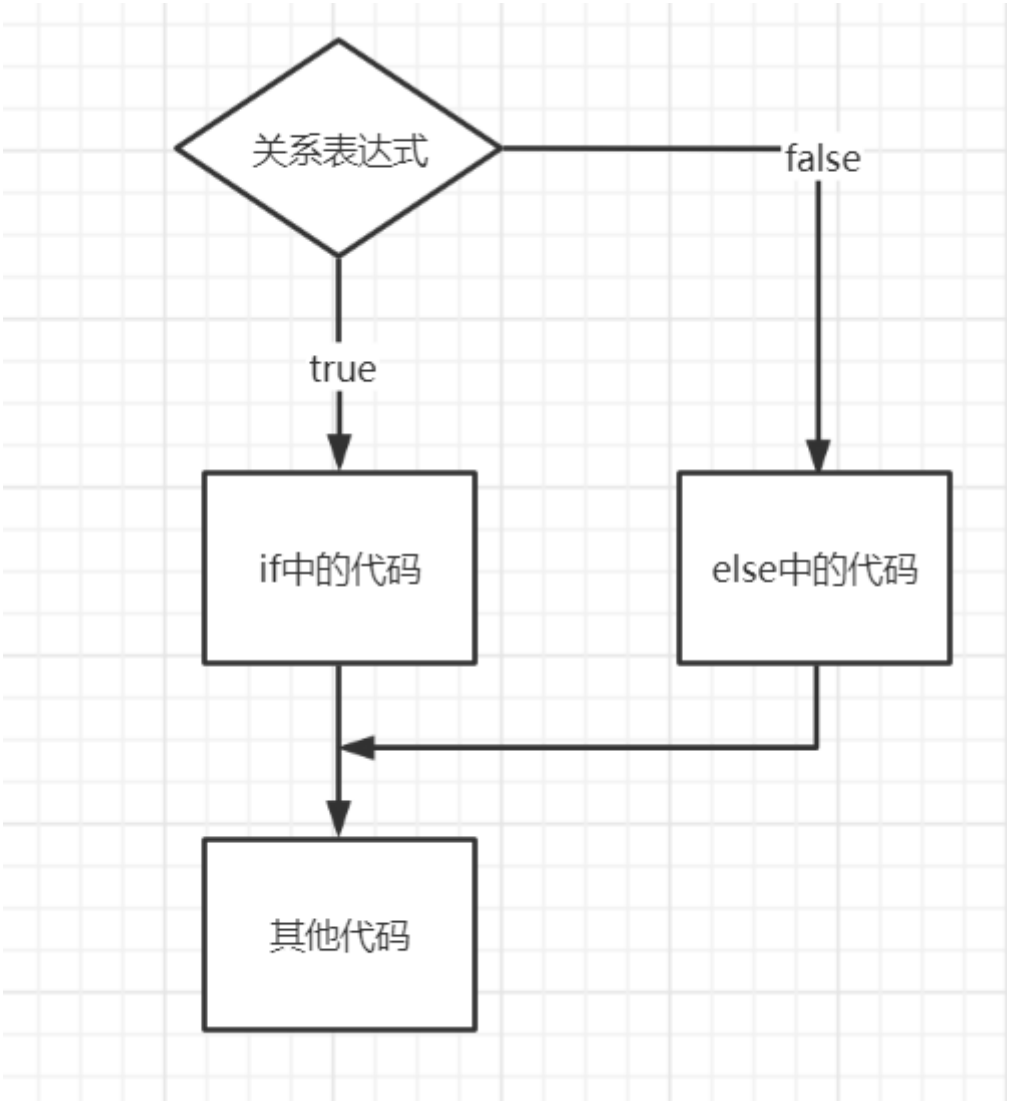
语法2：

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

执行流程：

1. 首先计算 **关系表达式** 的值
2. 如果关系表达式的值为true，就执行语句体1
3. 如果关系表达式的值为false，就执行语句体2
4. 继续执行if代码块后面的其他代码

如图：



例如，

```
int a = 10;
if(a%2==0){
    System.out.println("变量a的值为偶数");
}
else{
    System.out.println("变量a的值为奇数");
}
```

> if和else形成了一个组合，特点就是如果if中的代码执行了，那么else的代码就不执行，如果if中的代码没执行，那么else中的代码就会执行。也就是if和else这俩个语句中的代码，【一定】是有唯一的一个执行，而另一个不执行。

但是，如果有俩个if，那么它们俩个是相互独立切互不影响的俩个结构。

例如，

```
int a = 10;
if(a%2==0){
    System.out.println("变量a的值为偶数");
}
if(a%2==1){
    System.out.println("变量a的值为奇数");
}
```

> 注意，第一个if条件无论是true还是false，第二个if条都会继续判断，这个逻辑和if-else是不同的

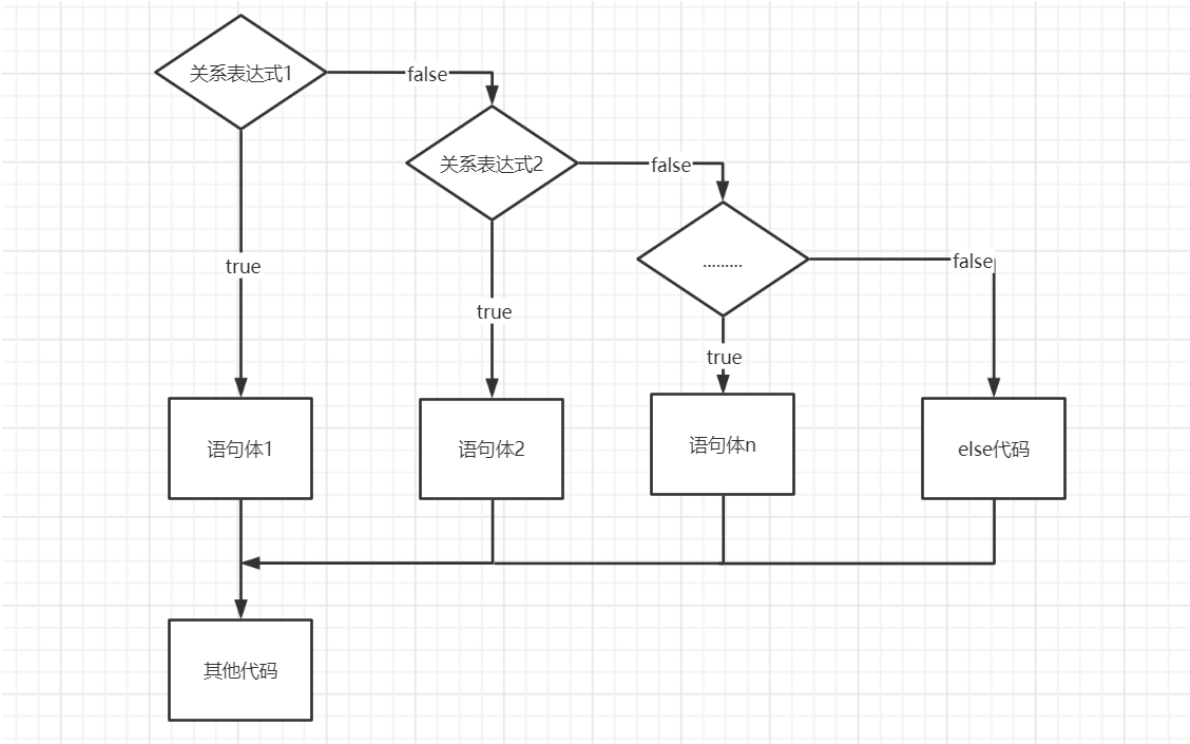
语法3：

```
if (关系表达式1) {
    语句体1;
}
else if (关系表达式2) {
    语句体2;
}
…
else {
    //else代码;
}
//其他代码
```

执行流程：

1. 首先计算 **关系表达式1** 的值
2. 如果表达式1的值为true，就执行语句体1；如果值为false就计算**关系表达式2**的值
3. 如果表达式2的值为true，就执行语句体2；如果值为false就计算**关系表达式3**的值
4. ....
5. 如果没有任何关系表达式为true，就执行else代码
6. 如果中间有任何一个关系表达式为true，那么执行完对应的代码语句之后，整个if-elseif-else退出

如图：



例如，

```
int a = 10;
if(a>90){
    System.out.println("优秀");
}
else if(a>80){
    System.out.println("良好");
}
else if(a>70){
    System.out.println("中等");
}
else if(a>60){
    System.out.println("及格");
}
else{
    System.out.println("不及格");
}
```

> 从上到下依次判断，有一个判断为true执行了代码，那么后续判断都不再执行，如果判断都为false，则执行else语句代码

例如：实现一个方法，方法调用完后会返回一个问候的语句，如果是8点~12点之间，那么就返回早上好，如果是12点~14点，则返回中午好，如果是14点~18点，则返回下午好，其他情况，返回晚上好。

```
public String sayHello(int hour){
    String message;

    if(hour>=8 && hour<12){
        message = "早上好";
    }
    else if(hour>=12 && hour<14){
```

```
            message = "中午好";
        }
        else if(hour>=14 && hour<18){
            message = "下午好";
        }
        else{
            message = "晚上好";
        }

        return message;
    }
```

例如：实现一个方法，方法需要传一个参数，表示年份，方法调用完后会返回一个boolean值，表示这个年份是不是闰年。

闰年判断标准：以下条件满足一个，就是闰年

- 能被4整除，但是不能被100整除
- 能被400整除

```
    public boolean isLeapYear(int year){
        boolean flag = false;

        if((year%4==0 && year%100!=0) || year%400==0){
            flag = true;
        }

        return flag;
    }

    //这俩方法中的代码是等价的

    public boolean isLeapYear(int year){
        return (year%4==0 && year%100!=0) || year%400==0;
    }


```

4.2 switch

switch语句和if很类似，都是用来判断值是否相等，但是switch默认只支持byte、short、int、char这四种类型的比较，JDK8中也允许String类型的变量做对比。

> 注意，使用switch来完成的功能，同样可以使用if来完成，但是使用if完成的功能，使用switch不一定能完成。

语法：

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

执行流程：

1. 首先计算出表达式的值
2. 其次，和case依次比较，一旦有对应的值，就会执行相应的语句，遇到break就会结束switch程序
3. 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

例如，

```
int mode = 0;//0 1 2 3

switch(mode){
    case 0:{
        System.out.println("默认模式开启");
        break;
    }

    case 1:{
        System.out.println("手动模式开启");
        break;
    }

    case 2:{
        System.out.println("自动模式开启");
        break;
    }

    case 3:{
        System.out.println("智能模式开启");
        break;
    }

    default:{
        System.out.println("模式选择错误!!");
    }

}
```

> 假如mode本次的值是0，那么case 0 这种情况就成立了，然后打印指定语句，再执行break，接着退出整个switch语句。

> 也就是说case 1 2 3 default这几种情况的代码就不再判断也不再执行。这一切都是因为执行了break。

如果没写break，那么这时候就会变成另外一种情况：

```
int mode = 0;//0 1 2 3

switch(mode){

    case 0:{
        System.out.println("默认模式开启");
    }

    case 1:{
        System.out.println("手动模式开启");
    }

    case 2:{
        System.out.println("自动模式开启");
    }

    case 3:{
        System.out.println("智能模式开启");
    }

    default:{
        System.out.println("模式选择错误!!");
    }

}
```

> 这个代码中的break全都去掉了
>
> 假设本次mode的值还是0，那么case 0成立之后，现在执行里面的代码， 打印指定语句。
>
> 由于这时候没有break，然后代码会继续往下执行，并且不会再做case 1 2 3的判断了，而是直接执行case 1 2 3中的代码，也包含执行default中的代码，所以最后的输出结果为：
>
> 默认模式开启
> 手动模式开启
> 自动模式开启
> 智能模式开启
> 模式选择错误!!
>
> 这种情况，就是因为代码中没有写break的原因。

例如，实现一个方法，方法需要一个int类型参数，方法中可以把这个数字转换为对应的星期几，例如 0 对应星期天，1对应星期一，方法最后需要把转换的结果返回。

```
public String getDayStr(int day){
    String result;

    switch(day){
        case 0:{
```

```
            result = "星期天";
            break;
        }

        case 1:{
            result = "星期一";
            break;
        }

        case 2:{
            result = "星期二";
            break;
        }

        case 3:{
            result = "星期三";
            break;
        }

        case 4:{
            result = "星期四";
            break;
        }

        case 5:{
            result = "星期五";
            break;
        }

        case 6:{
            result = "星期六";
            break;
        }

        default:{
            result = "参数有误，参数day的值可以在[0,6]之间";
        }
    }

    return result;
}
```

例如，实现一个方法，方法需要俩个int类型参数，一个表示年份，一个表示月份，方法的返回值也是int，返回值的含义是指定年份指定月份一共有多少天

```
public int getDayOfMonth(int year,int month){
    IfTest t = new IfTest();
    int num = 31;

    switch(month){
        case 4:
        case 6:
        case 9:
        case 11:{
            num = 30;
            break;
```

```
            }
            case 2:{
                num = ( t.isLeapYear(year)?29:28 );
            }
        }

        return num;
    }
```

> 注意，isLeapYear这方法是调用的IfTest类中之前写好的方法

4.3 for

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。

语法：

```
for (初始化语句;条件判断语句;条件控制语句) {
    循环体语句;
}
```

描述：

1. 初始化语句： 用于表示循环开启时的起始状态，简单说就是循环开始的时候什么样
2. 条件判断语句：用于表示循环反复执行的条件，简单说就是判断循环是否能一直执行下去
3. 循环体语句： 用于表示循环反复执行的内容，简单说就是循环反复执行的事情
4. 条件控制语句：用于表示循环执行中每次变化的内容，简单说就是控制循环是否能执行下去

执行流程：

1. 执行初始化语句
2. 执行条件判断语句，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
3. 执行循环体语句
4. 执行条件控制语句
5. 回到步骤2继续执行

例如，在控制台输出1-5和5-1的数据

```java
public class ForTest01 {
    public static void main(String[] args) {
        //需求：输出数据1-5
        for(int i=1; i<=5; i++) {
            System.out.println(i);
        }
        System.out.println("--------");
        //需求：输出数据5-1
        for(int i=5; i>=1; i--) {
            System.out.println(i);
        }
    }
}
```

例如，求1-5之间的数据和，并把求和结果在控制台输出

```java
public class ForTest02 {
    public static void main(String[] args) {
        //求和的最终结果必须保存起来，需要定义一个变量，用于保存求和的结果，初始值为0
        int sum = 0;
        //从1开始到5结束的数据，使用循环结构完成
        for(int i=1; i<=5; i++) {
            //将反复进行的事情写入循环结构内部
             // 此处反复进行的事情是将数据 i 加到用于保存最终求和的变量 sum 中
            sum += i;
            /*
                sum += i;   sum = sum + i;
                第一次：sum = sum + i = 0 + 1 = 1;
                第二次：sum = sum + i = 1 + 2 = 3;
                第三次：sum = sum + i = 3 + 3 = 6;
                第四次：sum = sum + i = 6 + 4 = 10;
                第五次：sum = sum + i = 10 + 5 = 15;
            */
        }
        //当循环执行完毕时，将最终数据打印出来
        System.out.println("1-5之间的数据和是：" + sum);
    }
}
```

例如，求1-100之间的偶数和

```java
public class ForTest03 {
    public static void main(String[] args) {
        //求和的最终结果必须保存起来，需要定义一个变量，用于保存求和的结果，初始值为0
        int sum = 0;
        //对1-100的数据求和与1-5的数据求和几乎完全一样，仅仅是结束条件不同
        for(int i=1; i<=100; i++) {
            //对1-100的偶数求和，需要对求和操作添加限制条件，判断是否是偶数
            if(i%2 == 0) {
                sum += i;
            }
        }
        //当循环执行完毕时，将最终数据打印出来
        System.out.println("1-100之间的偶数和是：" + sum);
```

```
        }
    }
```

思考，是否还有其他方式，可以实现1-100之间的偶数和

for循环的其他形式

例如，

```
    //for循环的常用写法
    for(int i=0;i<10;i++){
        System.out.println("hello world");
    }

    //初始化变量和改变变量的值，是可以写到其他地方的
    //这个最后的效果和上面的for循环是一样的
    int i = 0;
    for(;i<10;){
        System.out.println("hello world");
        i++;
    }

```

使用for循环编写死循环

例如，

```
    //这是一个死循环代码,for的小括号中，只有俩个分号
    for(;;){
        System.out.println("hello world");
    }

    //for循环的大括号中，如果只有一句代码，那么可以把大括号省去不写
    for(;;) System.out.println("hello world");
```

思考，一般情况下我们不会编写死循环代码，但是死循环是否有它专门的应用场景呢？

4.4 while

语法：

```
    初始化语句;
    while (条件判断语句) {
        循环体语句;
        条件控制语句;
    }
```

执行流程：

1. 执行初始化语句

2. 执行条件判断语句，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
3. 执行循环体语句
4. 执行条件控制语句
5. 回到步骤2继续

例如，循环不断的生成[0,9]随机数，直到生成随机数为5的时候，那么就停止这个循环。

```
int num = -1;
while(num!=5){
    num = (int)(Math.random()*10);
    System.out.println("num = "+num);
}
```

> 注意，Math是java.lang包的，可以直接使用，而不需要import导入
>
> Math.random()方法会返回[0,1)之间的随机数，返回值类型是double

例如，使用for循环完成上述功能

```
int num = -1;
for(;num!=5;){
    num = (int)(Math.random()*10);
    System.out.println("num = "+num);
}
```

例如，世界最高山峰是珠穆朗玛峰(8844.43米=8844430毫米)，假如有一张足够大的纸，它的厚度是0.1毫米。那么折叠多少次，可以折成珠穆朗玛峰的高度

```
public class WhileTest {
    public static void main(String[] args) {
        //定义一个计数器，初始值为0
        int count = 0;
        //定义纸张厚度
        double paper = 0.1;
        //定义珠穆朗玛峰的高度
        int zf = 8844430;
        //因为要反复折叠，所以要使用循环，但是不知道折叠多少次，这种情况下更适合使用while循环
        //折叠的过程中当纸张厚度大于珠峰就停止了，因此继续执行的要求是纸张厚度小于珠峰高度
        while(paper <= zf) {
            //循环的执行过程中每次纸张折叠，纸张的厚度要加倍
            paper *= 2;
            //在循环中执行累加，对应折叠了多少次
            count++;
        }
        //打印计数器的值
        System.out.println("需要折叠：" + count + "次");
    }
}
```

使用while循环编写死循环

例如，

```
while(true){
    //循环体代码
}
```

4.5 do-while

do-while循环和while循环很类似，只是do-while循环需要先执行循环体中的代码，然后再进行条件判断，是否可以进行一下次循环。其特点是，无论如何都会先执行一次大括号中的代码

语法：

```
初始化语句;
do {
    循环体语句;
    条件控制语句;
}while(条件判断语句);
```

执行流程：

1. 执行初始化语句
2. 执行循环体语句
3. 执行条件控制语句
4. 执行条件判断语句
    - 如果是false，循环结束
    - 如果是true，继续执行
5. 回到步骤2继续

例如，循环不断的生成[0,9]随机数，直到生成随机数为5的时候，那么就停止这个循环。

```
int a;
do{
    a = (int)(Math.random()*10);
    System.out.println("a = "+a);
}while(a!=5);
```

使用do-while循环编写死循环

例如，

```
do{
    //循环体代码
}while(true);
```

4.6 区别

三种循环的区别：

1、for、while 与 do ... while 的区别

- for和while 先判断条件是否成立，然后决定是否执行循环体（先判断后执行）
- do...while 先执行一次循环体，然后判断条件是否成立，是否继续执行循环体（先执行后判断）

2、for 与 while 的区别

- 条件控制语句所控制的自增变量，因为**默认情况下**归属for循环的语法结构中，在for循环结束后，就不能再次被访问到了

```
for(int i=0;i<10i++){
    //...
}
//这里无法使用变量i
```

- 条件控制语句所控制的自增变量，对于while循环来说不归属其语法结构中，在while循环结束后，该变量还可以继续使用

```
int i = 0;
while(i<10){
    //...
    i++;
}
//这里可以继续使用变量i
```

三种循环的使用场景：

1、明确循环次数，推荐使用for

2、不明确循环次数，推荐使用while

3、do..while 很少使用

4.7 循环嵌套

> 在一个循环中，可以嵌套另一个循环。

例如，输出5个空行

```
for(int i=0;i<5;i++){
    System.out.println();
}
```

例如，在同一行，输出10个五角星

```
for(int i=0;i<10;i++){
    System.out.print("☆");
}
```

例如，输出5行，每行10个五角星

```
for(int i=0;i<5;i++){
    for(int j=0;j<10;j++){
        System.out.print("☆");
    }
    System.out.println();
}
```

> println方法最后会自动换行
> print 方法最后不会自动换行

例如，输出以下内容：

```
*
**
***
****
*****
```

```
for(int i=0;i<5;i++){
    for(int j=0;j<i+1;j++){
        System.out.print("*");
    }
    System.out.println();
}
```

例如，输出以下内容：

```
    *
   ***
  *****
 *******
*********
```

```
//参数line表示要输出的行数
public void test(int line){
    //外层循环控制打印的行数
    for(int i=1;i<=line;i++){
        //这个循环控制每行打印的空格
        for(int j=0;j<line-i;j++){
            System.out.print(" ");
        }
        //这个循环控制每行打印的*
        for(int k=0;k<(2*i-1);k++){
            System.out.print("*");
        }
        //当前行中的空格和*都打印完了，最后输出一个换行
        System.out.println();
```

```
        }

    }
```

例如，输出以下内容：

```
***********
 *********
  *******
   *****
    ***
     *
```

```
    //参数line表示要输出的行数
    public void test(int line){
        //外层循环控制打印的行数
        for(int i=1;i<=line;i++){
            //这个循环控制每行打印的空格
            for(int j=0;j<(i-1);j++){
                System.out.print(" ");
            }
            //这个循环控制每行打印的*
            for(int k=0;k<(2*line-2*i+1);k++){
                System.out.print("*");
            }
            //当前行中的空格和*都打印完了，最后输出一个换行
            System.out.println();
        }

    }
```

5 break

> break 的意思是退出，结束当前的循环或switch代码。

例如，for循环从0到10进行输出，当i的值为5时，跳出当前循环（循环整体结束）

```
    for(int i=0;i<10;i++){
        System.out.println("i = "+i);
        if(i==5){
            break;
        }
    }
```

6 continue

> continue 的意思是结束本次循环，让循环直接进入一次的运行。

例如，for循环从0到10进行输出，当i的值为5时，结束本次循环，进入一下次循环

```
for(int i=0;i<10;i++){

    if(i==5){
        continue;
    }
    System.out.println("i = "+i);

}
```

7 label

例如，在嵌套循环中使用 break 或 continue 关键字

```
for(int i=0;i<3;i++){//外层循环
    for(int j=0;j<5;j++){//内层循环
        if(j==2){
            break;
        }
    }
}
```

> 注意，默认情况下，在嵌套循环中，break和continue只能默认对当前循环起作用。

如果想让break或continue针对某一个指定的循环起作用，那么可以使用label标签给循环起名字，然后使用break或continue加上循环的名字即可。

例如，

```
test1:for(int i=0;i<3;i++){//外层循环
    test2:for(int j=0;j<5;j++){//内层循环
        if(j==2){
            break test1;
        }
        System.out.println("i="+i+",j="+j);
    }
    System.out.println("----------------------");
}
```

例如，程序运行后，用户可多次查询星期对应的减肥计划，直到输入0，程序结束

提示，`java.util.Scanner`类，可以接收用户从键盘中的输入，该类是Java提供好的API，我们在代码中导入后可以直接使用。

```
 import java.util.Scanner;

 public class Test {
```

```
4
5       //不明确用户操作几次，使用死循环包裹业务逻辑
6       //给循环添加一个label标签
7       //匹配到0的时候，使用break结束循环死循环
8       public static void main (String[] args){
9
10          lo:while(true){
11              System.out.println("请输入您要查看的星期数:");
12              System.out.println("(如无需继续查看,请输入0退出程序)");
13
14              // 1. 键盘录入星期数据，使用变量接收
15              Scanner sc = new Scanner(System.in);
16              int week = sc.nextInt();
17
18              // 2. 多情况判断，采用switch语句实现
19              switch(week){
20                  // 3. 在不同的case中，输出对应的减肥计划
21                  case 0:
22                      System.out.println("感谢您的使用");
23                      break lo;
24                  case 1:
25                      System.out.println("跑步");
26                      break;
27                  case 2:
28                      System.out.println("游泳");
29                      break;
30                  case 3:
31                      System.out.println("慢走");
32                      break;
33                  case 4:
34                      System.out.println("动感单车");
35                      break;
36                  case 5:
37                      System.out.println("拳击");
38                      break;
39                  case 6:
40                      System.out.println("爬山");
41                      break;
42                  case 7:
43                      System.out.println("好好吃一顿");
44                      break;
45                  default:
46                      System.out.println("您的输入有误");
47                      break;
48              }
49          }
50
51
52      }
53  }
```

8 Random

Random类似Scanner，也是Java提供好的API，内部提供了产生随机数的功能，在这里可以先简单的使用下，后续的学习中会了解到更多详细细节。

例如，产生随机数1-10之间的

```
import java.util.Random;

public class Demo1Random {

    /*
        Random : 产生随机数

        1. 导包   : import java.util.Random;
                    导包的动作必须出现在类定义的上面

        2. 创建对象 : Random r = new Random();
                    上面这个格式里面，r 是变量名，可以变，其他的都不允许变

        3. 获取随机数 : int number = r.nextInt(10);  //获取数据的范围：[0,10) 包括0,不包括10
                    上面这个格式里面，number是变量名，可以变，数字10可以变。其他的都不允许变

        需求：产生随机数1-10之间的
    */
    public static void main(String[] args){

        Random r = new Random();

        for(int i = 1; i <= 10; i++){

            int num = r.nextInt(10) + 1;        // 1-10
            System.out.println(num);
        }

    }
}
```

例如，完成猜数字游戏

程序自动生成一个1-100之间的数字，使用程序实现猜出这个数字是多少

当猜错的时候根据不同情况给出相应的提示

1. 如果猜的数字比真实数字大，提示你猜的数据大了
2. 如果猜的数字比真实数字小，提示你猜的数据小了
3. 如果猜的数字与真实数字相等，提示恭喜你猜中了

例如，

```
import java.util.Scanner;
import java.util.Random;

public class Demo2Random {
    /*
        1. 准备Random和Scanner对象，分别用于产生随机数和键盘录入
        2. 使用Random产生一个1-100之间的数，作为要猜的数
```

```
        3. 键盘录入用户猜的的数据
        4. 使用录入的数据(用户猜的数据)和随机数(要猜的数据)进行比较, 并给出提示

        5. 以上内容需要多次进行, 但无法预估用户输入几次可以猜测正确, 使用while(true)死循环包裹
        6. 猜对之后, break结束.
    */
    public static void main(String[] args){
        // 1. 准备Random和Scanner对象, 分别用于产生随机数和键盘录入
        Random r = new Random();
        Scanner sc = new Scanner(System.in);

        // 2. 使用Random产生一个1-100之间的数, 作为要猜的数
        int randomNum = r.nextInt(100) + 1;

        // 5. 以上内容需要多次进行, 但无法预估用户输入几次可以猜测正确, 使用while(true)死循环包裹
        while(true){
            // 3. 键盘录入用户猜的的数据
            System.out.println("请输入您猜的数据:");
            int num = sc.nextInt();
            // 4. 使用录入的数据(用户猜的数据)和随机数(要猜的数据)进行比较, 并给出提示
            if(num > randomNum){
                System.out.println("猜大了");
            }else if(num < randomNum){
                System.out.println("猜小了");
            }else{
                // 6. 猜对之后, break结束.
                System.out.println("恭喜,猜中了");
                break;
            }
        }

        System.out.println("感谢您的使用");

    }
}
```