Java 入门- 预科班内容正课会重讲

内参保密不得外传

达内 Java 培优方向

陈子枢 主讲

今日头条极速版

2020

1 第四天：流程控制：分支+循环+集合+遍历

1.1 流程控制

1.1.1 程序的三种基本结构

- 顺序结构：代码依次从上往下执行
- 分支结构：根据判断，有些代码执行，有些代码不执行
- 循环结构：反复执行一段代码

1.2 分支结构：if

1.2.1 单分支

例如：天晴晾衣服，下雨收衣服

```
public static void main(String[] args) {
    boolean isRainy = true;
    boolean isSunny = true;

    if(isRainy) {
        System.out.println("下雨啦，收衣服啦");
        isSunny = false;
    }

    if(isSunny) {
        System.out.println("谁说下雨啦，大太阳金光灿烂的");
    }
}
```

1.2.2 多分支

例如：60 分以下不及格，61~70 及格，71~80 中等，81~90 良好，91~100 优秀

```
package javase.base.ifc;

public class TestIfScore {
    public static void main(String[] args) {
        TestIfScore ts = new TestIfScore();

        int score  = 100;
        System.out.println("成绩: "+ts.degree(score));
    }

    // 60 分以下不及格，61~70 及格，71~80 中等，81~90 良好，91~100 优秀
    public String degree(int score) {
        String result = "";

        if (score > 0 && score <= 60) {
            result = "不及格";
        } else if (score > 60 && score <= 70) {
            result = "及格";
        } else if (score > 70 && score <= 80) {
            result = "中等";
        } else if (score > 80 && score <= 90) {
            result = "良好";
        } else if (score > 90 && score <= 100) {
            result = "优秀";
        } else {
            result = "数据非法! " + score;
        }
        return result;
    }
}
```

上面这种方式实际开发中使用非常频繁。

```
    @Test   //需求: 分数 90-100 优秀，80-90 良好，70-80 一般，60-70 及格，60
以下不及格
    public void testScore() {
        int score = 60;
        if( score>=90 ) {
            System.out.println("成绩优秀");
        }else if( score>=80 ) {
            System.out.println("成绩良好");
        }else if( score>=70 ) {
            System.out.println("成绩一般");
        }else if( score>=60 ) {
```

```
            System.out.println("成绩及格");
        }else {
            System.out.println("成绩不及格");
        }
    }
```

上面是标准写法，而下面这个判断条件只是一半，为什么也可以呢？这就是利用它的语法。If 为分支结构，一个条件成立其他的不再判断，每个条件是互斥的，非此即彼，所以不会发生交叉重复。

1.3 分支结构：switch

1.3.1 格式

switch(expr1)中，expr1 是一个整数表达式，整数表达式可以是 int 基本类型或 Integer 包装类型，由于 byte,short,char 都可以隐含转换为 int，所以也支持。实际开发中常用 string 类型，可竟然不支持，这么傻，真不可思议，其他语言天生就支持，最后 jdk 1.7 终于新增支持 String 了。

switch 了解即可，基本可以通过 if-else 来替代，而且 java 的 switch 很笨，远不如其他语言灵活，例如 vb。

```
switch(变量或者表达式){
    case 1:
    case 2:
    case 3:
    case 4:
    default:
}
```

1.3.2 例子：电话号码

```
    public static void main(String[] args) {
        int phone = 110;
        String msg = "";
        switch(phone) {
            case 110 :
                msg = "我是警察";
                break;
            case 120 :
                msg = "我是医生";
                break;
            case 119 :
                msg = "我是火警";
                break;
```

```
            default :
                msg = "错误电话号码";
        }
        System.out.println(msg);
    }
```

1.3.3 例子：ATM 机

```
    public static void main(String[] args) {
        System.out.println("请选择服务项目：");
        System.out.println("1.存款");
        System.out.println("2.取款");
        System.out.println("3.查询");
        System.out.println("4.退出");

        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        switch (n) {
        case 1:
            System.out.println("请输入存款金额");
            break;
        case 2:
            System.out.println("请输入取款金额");
            break;
        case 3:
            System.out.println("您的账户里还有 100 万人民币");
            break;
        case 4:
            return;
        default:
            System.out.println("输入错误，请重新输入");
            break;
        }
    }
```

1.3.4 例子：学生成绩

```
package cn.tedu.pojo;

public class TestSwitchScore {
    public static void main(String[] args) {
        //switch()里面的参数值不能是一个范围，是一个具体值（int,String)
        //switch 语句非常死板，开发中用 if-else if 替代

        //需求：一组分数，判断等级，
```

```
        //小于 60 不及格，60~70 及格，70~80 一般，80~90 良好，90~100 优秀
        int score = 34;

        //System.out.println( score/10 );       //技巧：利用整除方式
        switch( score/10 ) {
        case 10 :
            System.out.println("优秀");
            break;
        case 9 :
            System.out.println("优秀");
            break;
        case 8 :
            System.out.println("良好");
            break;
        case 7 :
            System.out.println("一般");
            break;
        case 6 :
            System.out.println("及格");
            break;
        default :
            System.out.println("不及格");
        }
    }
}
```

1.4 循环结构：for

1.4.1 概念

循环结构是实际开发非常常见的用法，它是指在程序中需要反复执行某个功能而设置的一种程序结构。它由循环体中的条件，判断继续执行某个功能还是退出循环。注意，必须要有判断来结束循环，否则就死循环了。

1.4.2 打印 0 到 9

正向 i++，反向 i--

```
    public static void main(String[] args) {
        for(int i=0; i<10; i++) {
            System.out.println(i);
        }
    }
```

1.4.3 反向循环

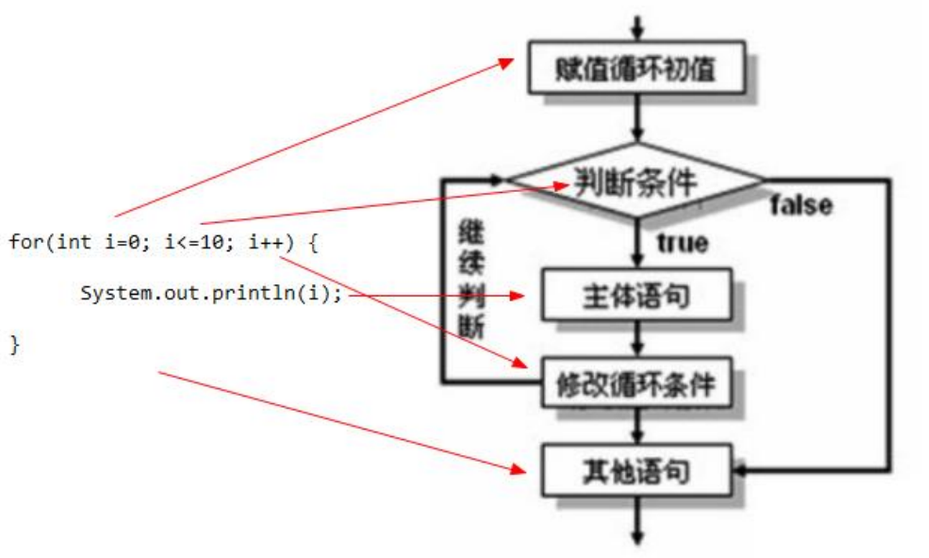
```
public static void main(String[] args) {
    for(int i=9; i>-1; i--) {
        System.out.println(i);
    }
}
```

1.4.4 死循环

```
public static void main(String[] args) {
    for(int i=10; i>0; i++) {
        System.out.println(i);
    }
}
```

i>0 判断条件永远成立，循环体内代码就永远执行。可以看到判断条件有多重要，判断错误就形成了死循环，永不停歇，只到内存耗尽，报错。

1.4.5 流程图



1.4.6 执行顺序

下面的代码我们执行过，很多同学觉得应该打印结果是 1~10 啊，为何呢？先执行判断啊，然后执行 i++，最后打印 i++的值，当然应该 i 加过了。可实际为何是 0~9 呢？

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

```
        System.out.println(i);
    }
```

打印结果：0~9

我们修改下，大家看看结果什么样？

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

打印结果：1~10，这会和我们想的一样了吧。那为何上面的不是呢？是我们有个误区，这里 for 循环内部的结构是先执行方法体，后执行 i++哦。怎么证明呢？上面代码也是猜测啊，很简单可以通过我们所学的断点跟踪来验证。

```
    package javase.base.circle;

    public class TestFor {
        public static void main(String[] args) throws Exception {
            for(
                    int i=0;          //第一步执行
                    i<7;              //第二步执行
                    i++               //第四步执行
                ) {
                System.out.println(i);    //第三步执行
            }
        }
    }
```

1.4.7 获取集合元素

这是实际开发中最常见的一种方式

```
        public static void main(String[] args) {
            //声明一个整形数组，并且初始化成绩
            int[] scores = new int[] {100, 99, 98, 80, 60, 30};

            //length 为数组的一个方法，获取整个数组的长度
            for(int i = 0; i< scores.length; i++) {
                //通过 x[i]访问对应下标的数组元素值
                System.out.println(scores[i]);
            }
        }
```

1.5 双重循环

1.5.1 矩形

双重循环，习惯变量声明为：i、j、k、m、n

```
@Test
public void Box() {
    for(int i = 0; i<5; i++) {
        for(int j=0; j<14; j++) {
            System.out.print("*");
        }
        System.out.println();
    }
}
```

1.5.2 二维数组遍历

```
@Test
public void twoArray() {
    int[][] scores = new int[2][3];      //定义二维数组

    System.out.println("一维长度："+ scores.length);
    System.out.println("二维长度："+ scores[0].length);

    //tony 语文100，数学99，英语60
    scores[0][0] = 100;
    scores[0][1] = 99;
    scores[0][2] = 60;

    //tina 语文88，数学60，英语95
    scores[1][0] = 88;
    scores[1][1] = 60;
    scores[1][2] = 95;

    //直接赋值方式
//      int[][] scores = new int[][]{{1,2,3}, {4,5,6}};
//     int[][] scores = {{1,2,3}, {4,5,6}};

    System.out.println("\n遍历二维数组：");
    for(int i=0; i< scores.length; i++) {
        for(int j=0; j< scores[0].length; j++) {
            System.out.print( scores[i][j] + ", ");
        }
        System.out.println();  //换行
    }
```

```
    }
```

1.6 循环结构：while

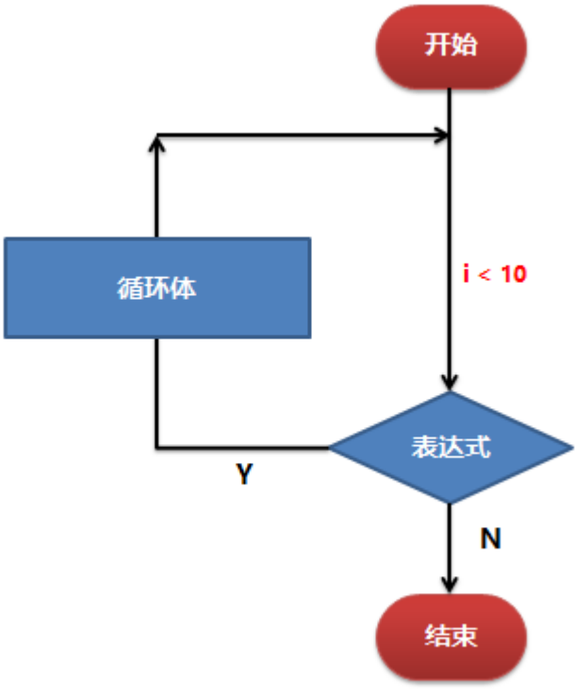
1.6.1 打印 0 到 9

```
        int i = 0;
        while (i<10) {
            System.out.println(i++);
        }
```

1.6.2 流程图

一图胜百文，流程图广泛使用，工作中特别常见各种工作流程的图。但其组成特别简单一看就懂。

- 圆角方块：代表开始和结束，老外习惯用大的原点表示
- 长条方块：代表多行代码
- 菱形：代表判断，Yes 或者 No，条件成立去哪，条件不成立去哪
- 箭头：代表了接下来的执行哪的代码



1.6.3 利用 break 退出死循环

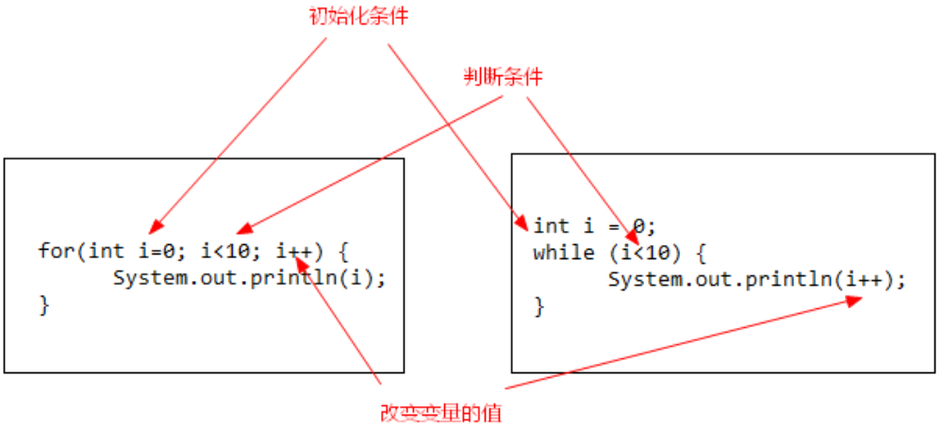
利用 break 语句退出循环，否则就死循环了

```
    public static void main(String[] args) {
        //这里是个死循环哦，但利用它我们可以一直获取控制台的数据
        while(true){
            String str = new Scanner(System.in).nextLine();

            //利用判断条件退出死循环
            if (str.equals("exit")) break;

            //遍历获取的用户输入内容
            for (int i = 0; i < str.length(); i++) {
                //挨个获取字符进行判断
                char c = str.charAt(i);
                if(c<='9'&&c>='0'){
                    System.out.print(c);
                }else {
                    //非 0 到 9 的数字则替换为*
                    System.out.print("*");
                }
            }

        }
    }
```

执行结果：

```
王菲生于 1969 年 8 月 8 日
*****1969*8*8*
谢霆锋生于 1980 年 8 月 29 日
******1980*8*29*
```

1.6.4 for 和 while 循环比较



可以看到两种方式可以互相替代,那开发中什么时候用 for 呢? 什么时候用 while 呢? 习惯上用 for 多些，for 一般已知数组的长度，而 while 循环可以支持不知道，后面就有例子。while(true) { … break; … }

1.7 循环结构：do-while

1.7.1 打印 0 到 9

```
package javase.base;

import org.junit.Test;

public class TestArea {

    @Test
    public void testWhile() {
        System.out.println("while loop");
        int i = 0;
        while (i<10) {
            System.out.println(i++);
        }
    }

    @Test
    public void testDo() {
        System.out.println("do-while loop");
        int i = 0;
        do {
```

```
            System.out.println(i++);
        } while (i<10);
    }

}
```

1.7.2 while 和 do-while 的差异

```
package javase.base;

import org.junit.Test;

public class TestArea {

    @Test
    public void testWhile() {
        System.out.println("while loop");
        int i = 0;
        while (i<0) {
            System.out.println(i++);
        }
    }

    @Test
    public void testDo() {
        System.out.println("do-while loop");
        int i = 0;
        do {
            System.out.println(i++);
        } while (i<0);
    }

}
```

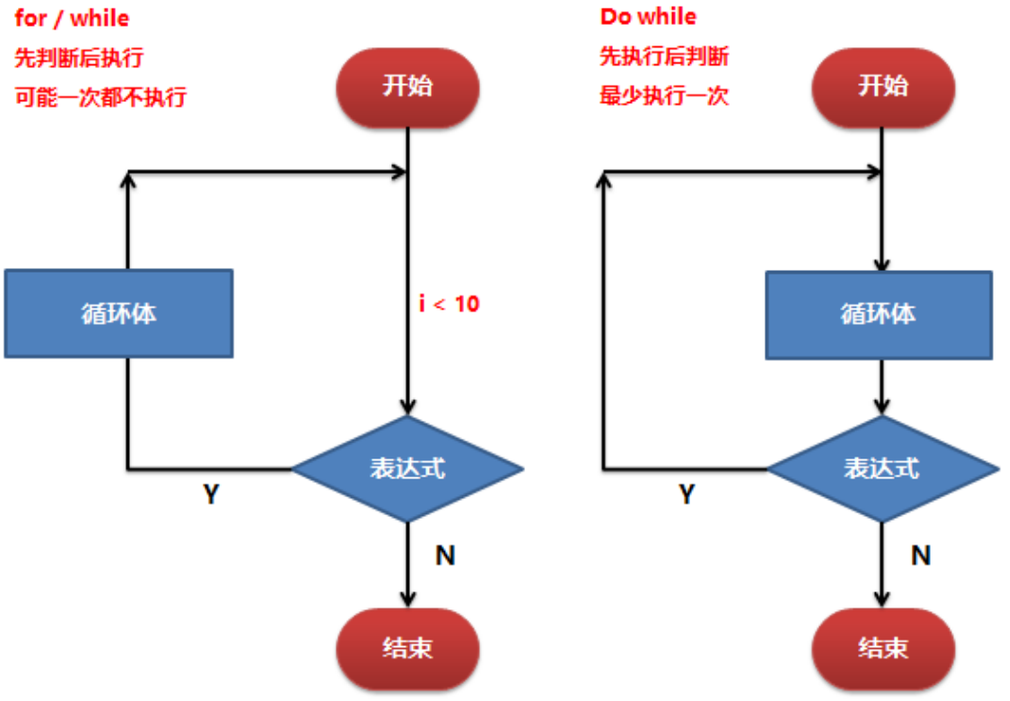
执行结果：

```
while loop
do-while loop
0
```

可以看到我仅仅把判断条件改了下和初始值相同，它们的区别立刻跃然纸上。While 没有执行一次，而 do 不论如何都会先执行一次。这个细微差别在实际业务中还是有的，当然这两种都没有 for 使用的频繁。

所以猛一下接受不了的同学，一会 for 循环，一会 while 循环，一会 do-while 循环，都蒙了。别着急，后面见多了，练多了自然就熟悉，目前可以放松要求，就学会 api 的写法，混个眼熟即可。

1.7.3 流程图



1.8 经典的死循环

1.8.1 for 死循环

```
for(;;) {
    System.out.println("*");
}
for(;ture;) {
    System.out.println("*");
}
```

1.8.2 while 死循环

```
while(true) {
    System.out.println("*");
}
```

1.9 集合 Collection

1.9.1 List

```
@Test
public void list() {
    //创建 list 对象，里面约定存储字符串元素
    List<String> courseList = new ArrayList<String>();
    courseList.add("语文");          //添加值
    courseList.add("数学");
    courseList.add("外语");

    courseList.set(0, "古文");       //修改值

    courseList.remove(2);        //删除元素

    System.out.println(courseList);

    for(String s : courseList) {     //遍历
        System.out.println(s);
    }
}
```

1.9.2 Map

```
@Test
public void map() {
    Map<String, String> map = new HashMap<>();
    map.put("800 年", "周天子");
    map.put("14 年", "秦始皇");
    map.put("407 年", "汉武帝");
    map.put("289", "唐太宗");
    map.put("296 年", "清太祖");

    //特点：key 值相同，内容自动覆盖，不会提示
    map.put("296 年", "唐玄宗");

    map.remove("14 年");                  //按 key 删除

    //根据名字遍历
    for(String key : map.keySet()) {
        System.out.println(key + " = " + map.get(key) );
    }
```

```
        //利用 entry 对象进行遍历
        for(Entry<String, String> entry : map.entrySet()) {
            System.out.println(entry.getKey() +" = "+entry.getValue());
        }
    }
```

1.9.3 Set

```
    @Test
    public void map() {
        Map<String, Object> map = new HashMap<>();
        map.put("800 年", "周天子");
        map.put("14 年", "秦始皇");
        map.put("407 年", "汉武帝");
        map.put("289", "唐太宗");
        map.put("296 年", "唐太宗");

        map.put("296 年", "唐玄宗");       //特点:key 值相同,内容自动覆盖,
不会提示

        map.remove("14 年");                  //按 key 删除

        for(String key : map.keySet()) {
            System.out.println(key + "=" + map.get(key) );
        }
    }
```

1.10集合遍历

1.10.1 For 循环遍历

```
        for(int i=0; i< courseList.size(); i++) {
            //根据下标获取元素
            System.out.println(courseList.get(i));
        }
```

1.10.2 Iterator 迭代器遍历

也叫增强 for 循环，数组下 for 循环效率高，链表下 iterator 效率高

```
        Iterator<String> it = courseList.iterator();
        while( it.hasNext() ) {
            String course = it.next();
            System.out.println(course);
```

```
        }
```

1.10.3 Foreach 遍历

foreach 遍历是 iterator 的简写，开发中对象遍历是最爱使用

```
        for(String s : courseList) {     //遍历
            System.out.println(s);
        }
```

1.10.4 源码

直接打印对象为何能看到对象内容,因为个集合类继承了抽象类 AbstractCollection，而这个抽象类重写了 toString 方法。

典型使用了 for 的死循环形式，使用 retrun 跳出并结束 for 循环，使用 StringBuilder 拼接字符串，使用迭代器遍历集合。

```
    public String toString() {
        Iterator<E> it = iterator();
        if (! it.hasNext())
            return "[]";

        StringBuilder sb = new StringBuilder();
        sb.append('[');
        for (;;) {
            E e = it.next();
            sb.append(e == this ? "(this Collection)" : e);
            if (! it.hasNext())
                return sb.append(']').toString();
            sb.append(',').append(' ');
        }
    }
```