第 7 天 Java 基础语法

今日内容介绍

- 循环练习
- 数组方法练习

第 1 章 循环练习

1.1 编写程序求 1+3+5+7+……+99 的和值。

题目分析：

通过观察发现，本题目要实现的奇数（范围 1-100 之间）的累加和。

1. 为了记录累加和的值，我们需要定义一个存储累加和的变量
2. 我们要获取到 1-100 范围内的数
3. 判断当前数是否为奇数，是奇数，完成累加和操作
4. 累加完毕后，最终显示下累加和的值

解题步骤：

1. 定义一个用来记录累加和的变量
2. 使用 for 循环语句，完成 1-100 之间每个数的获取
3. 使用 if 条件语句，判断当前数是否是奇数，是奇数，进行累加和操作
4. 使用输出语句，打印累加和变量的值

代码如下：

```
public class Test01 {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 0; i < 100; i++) {
            if (i%2==1) {
                sum += i;
            }
        }
        System.out.println("累加和的值 " + sum);
    }
}
```

1.2 输出所有的水仙花数，所谓水仙花数是指一个数 3 位数，其每位数字立方和等于其本身，如 153 = 1*1*1 + 3*3*3 + 5*5*5

题目分析：

通过观察发现，本题目要实现打印符合要求的数字（即水仙花数）。

1. 明确什么样的数就是水仙花数。水仙花数是指一个 3 位数（100-999 之间），其每位数字立方之和等于该 3 位数本身。如 153 = 1*1*1 + 3*3*3 + 5*5*5，

   即 3 位数本身 = 百位数立方 + 十位数立方 + 个位数立方;

2. 获取水仙花范围内的所有 3 位数（100-999 之间的每个 3 位数）

3. 判断该 3 位数是否满足水仙花数，满足，打印该 3 位数

解题步骤：

1. 使用 for 循环，得到 100-999 之间的每个 3 位数
2. 获取 3 位数中百位数字、十位数字、个位数字
3. 使用 if 条件语句，判断该 3 位数是否满足水仙花数，满足，使用输出语句，打印该 3 位数

代码如下：

```
public class Test02 {
```

```
        public static void main(String[] args) {
            for (int i = 100; i < 1000; i++) {
                int bai = i/100%10;
                int shi = i/10%10;
                int ge = i%10;

                if (i == bai*bai*bai + shi*shi*shi + ge*ge*ge) {
                    System.out.println(i);
                }
            }
        }
    }
```

1.3 ASCII 编码表

American Standard Code for Information Interchange，美国标准信息交换代码.

在计算机中，所有的数据在存储和运算时都要使用二进制数表示, a、b、c、d 这样的 52 个字母（包括大写）以及 0、1 等数字还有一些常用的符号，在计算机中存储时也要使用二进制数来表示，而具体用哪些二进制数字表示哪个符号，当然每个人都可以约定自己的一套（这就叫编码），而大家如果要想互相通信而不造成混乱，那么大家就必须使用相同的编码规则，于是美国有关的标准化组织就出台了 ASCII 编码，统一规定了上述常用符号用哪些二进制数来表示。

ASCII表

(American Standard Code for Information Interchange 美国标准信息交换代码)

高四位 / 低四位		ASCII控制字符 0000 0						0001 1						ASCII打印字符 0010 2		0011 3		0100 4		0101 5		0100 6		0111 7	
		十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	Ctrl
0000	0	0		^@	NUL	\0	空字符	16	▶	^P	DLE		数据链路转义	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH		标题开始	17	◀	^Q	DC1		设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☻	^B	STX		正文开始	18	↕	^R	DC2		设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX		正文结束	19	‼	^S	DC3		设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	♦	^D	EOT		传输结束	20	¶	^T	DC4		设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ		查询	21	§	^U	NAK		否定应答	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK		肯定应答	22	▬	^V	SYN		同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	•	^G	BEL	\a	响铃	23	↨	^W	ETB		传输块结束	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	8	◘	^H	BS	\b	退格	24	↑	^X	CAN		取消	40	(	56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	HT	\t	横向制表	25	↓	^Y	EM		介质结束	41	)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◙	^J	LF	\n	换行	26	→	^Z	SUB		替代	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	\v	纵向制表	27	←	^[	ESC	\e	溢出	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	12	♀	^L	FF	\f	换页	28	∟	^\	FS		文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|	
1101	D	13	♪	^M	CR	\r	回车	29	↔	^]	GS		组分隔符	45	-	61	=	77	M	93	]	109	m	125	}	
1110	E	14	♫	^N	SO		移出	30	▲	^^	RS		记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	15	☼	^O	SI		移入	31	▼	^-	US		单元分隔符	47	/	63	?	79	O	95	_	111	o	127	⌂	*Backspace 代码：DEL

数字 0-9 对应 ASCII 编码十进制为 48-57，字母 a-z 对应 ASCII 编码十进制为 97-122 ,字母 A-Z 对应 ASCII 编码十进制为 65-90

1.4 利用 for 循环打印 ABCDEFG...XYZ，26 个大写字母与 26 个小写字母

题目分析：

通过观察发现，本题目要实现打印 26 个大写字母、26 个小写字母

1. 一共 26 个大小写字母，那么，可以考虑循环 26 次。在每次循环中，完成指定字母的大小写打印
2. 找出 ABCDEFG...XYZ 这些字母之间的变化规律

通过 ASCII 表发现，后面的字母比它前面的字母，ASCII 值大 1

下一个字母 = 上一个字母 + 1

如： A B C D

65 66 67 68

3. 在每次循环中打印上一个字母大小写，并指定下一个字母

解题步骤：

1. 定义初始化大写变量，值为’A’； 初始化小写变量，值为’a’
2. 使用 for 循环，进行 26 次循环
3. 在每次循环中，打印大写字母、小写字母。

   每次打印完成后，更新大写字母值、小写字母值

代码如下：

```
public class Test04 {
    public static void main(String[] args) {
        char da = 'A';
        char xiao = 'a';
        for (int i = 0; i < 26; i++) {
            System.out.println("大写字母 "+da+" ,小写字母 "+xiao);
            da++; //更新大写字母值
            xiao++; //更新小写字母值
        }
    }
}
```

1.5 利用 for 循环打印 9*9 表?

如：

```
1*1=1
1*2=2  2*2=4
1*3=3  2*3=6  3*3=9
```

题目分析：

通过观察发现，如果把 1*1=1 这样的内容 看做一颗*的话，那么打印结果就成了如下效果：

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

…

这样，就是打印 9 行星，每行打印星的个数与当前行数相等。

再观察“1*3=3 2*3=6 3*3=9”得出它们如下的变化规律：

每行第n次 +"*"+ 行号 +"="+ 每行第n次 * 行号

```
如:  1   +"*"+  2   +"="+   1*2; // 相当于 1*2=2
     2   +"*"+  2   +"="+   2*2; // 相当于 2*2=4
```

解题步骤：

1. 定义一个外层 for 循环，初始值从 1 开始，循环 9 次。用来控制打印的行数
2. 在外层 for 循环内部，定义一个 for 循环，初始值从 1 开始，循环次数与当前行数相等。用来完成每行打印指定次数的乘法公式 如 1*1=1
3. 在内层 for 循环中，完成每行指定次数的乘法公式打印 如 1*1=1

```
System.out.print(k +"*"+ j +"="+ j*k +"\t");
// 变量 k 代表：每行中的第 n 次
// 变量 j 代表：行号
```

4. 在外循环中，当每行指定次数的乘法公式打印完毕后，通过 System.out.println()切换到下一行。这样，再次打印乘法公式时，就在下一行输出打印了

代码如下：

```
public class Test05 {
    public static void main(String[] args) {
        for (int j = 1; j < 10; j++) {
            for (int k = 1; k <= j; k++) {
                System.out.print(k +"*"+ j +"="+ j*k +"\t");
            }
            System.out.println();
        }
    }
}
```

第 2 章 数组方法练习

2.1 定义打印数组元素方法,按照给定的格式打印[11, 33, 44, 22, 55]

题目分析：

通过观察发现，本题目要实现按照指定格式，打印数组元素操作

1. 通过循环，我们可以完成数组中元素的获取，数组名[索引]
2. 观察发现，每个数组元素之间加入了一个逗号”,”进行分隔；并且，整个数组的前后有一对中括号”[]”包裹数组所有元素。

解题步骤：

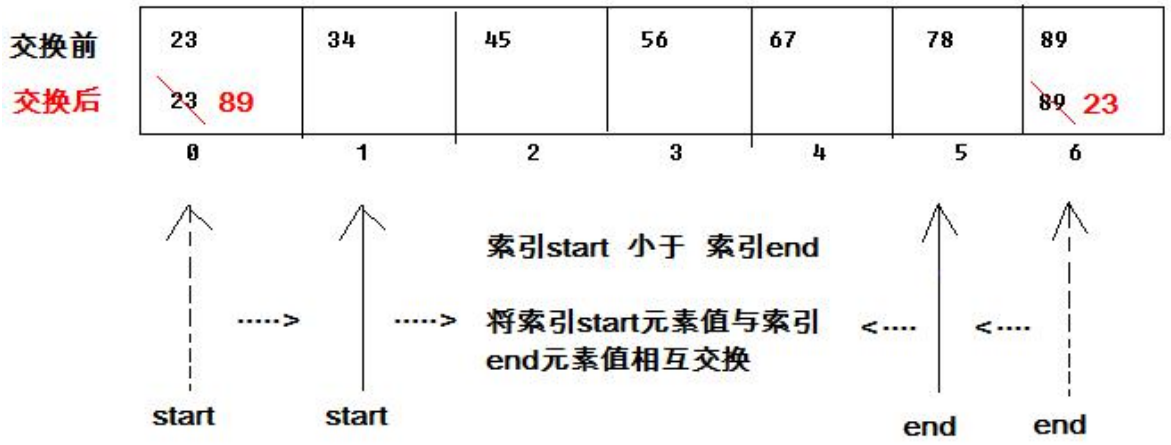
1. 使用输出语句完成打印 左边的中括号”[”
2. 使用循环，输出数组元素值。输出元素值分为两种情况，如下：
   a) 最后一个数组元素，加上一个右边的中括号”]”
   b) 非最后一个数组元素，加上一个逗号”,”

代码如下：

```
//打印数组
public static void printArray(int[] arr) {
	System.out.print("[");
	for (int i = 0; i < arr.length; i++) {
		if (i == arr.length - 1) {
			System.out.println(arr[i]+"]");
		} else {
			System.out.print(arr[i]+", ");
		}
	}
}
```

2.2 数组元素逆序

图解：



题目分析：

通过观察发现，本题目要实现原数组元素倒序存放操作。即原数组存储元素为{11,22,33,44}，逆序后为原数组存储元素变为{44,33,22,11}。

1. 通过图解发现，想完成数组元素逆序，其实就是把数组中索引为 start 与 end 的元素进行互换。
2. 每次互换后，start 索引位置后移，end 索引位置前移，再进行互换
3. 直到 start 位置超越了 end 位置，互换结束，此时，数组元素逆序完成。

解题步骤：

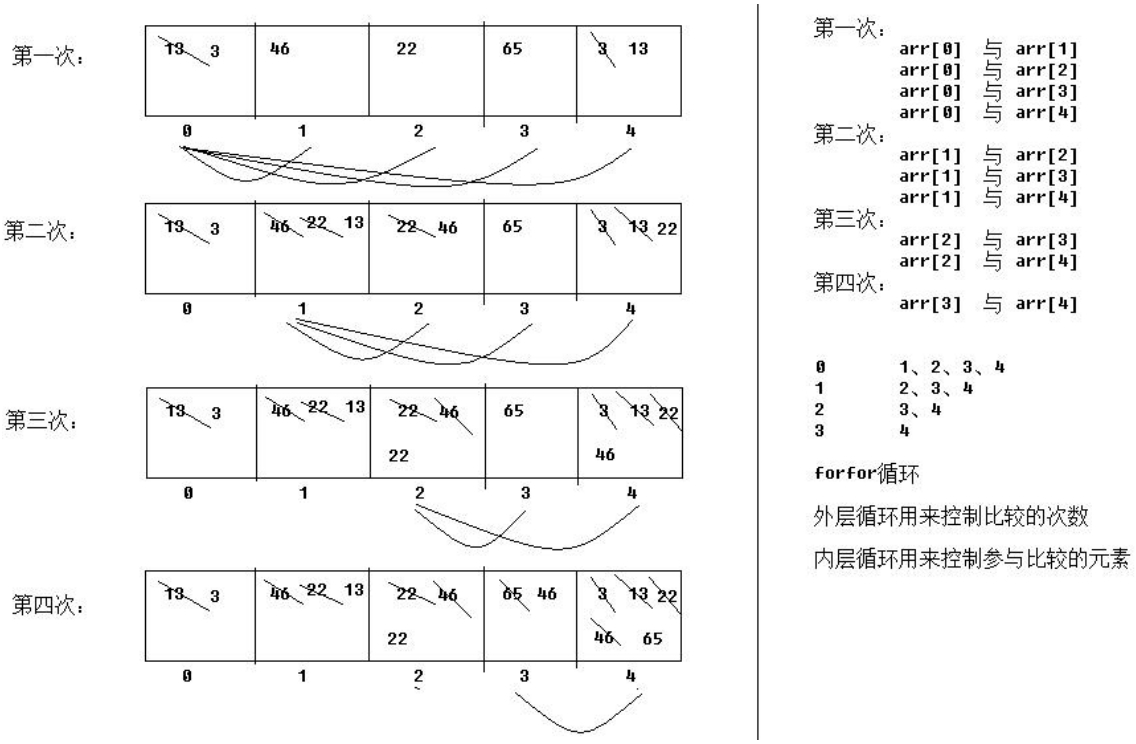
1. 定义两个索引变量 start 值为 0，变量 end 值为数组长度减去 1（即数组最后一个元素索引）
2. 使用循环，完成数组索引 start 位置元素与 end 位置元素值互换。
3. 在循环换过程中，每次互换结束后，start 位置后移 1，end 位置前移 1
4. 在循环换过程中，最先判断 start 位置是否超越了 end 位置，若已超越，则跳出循环

代码如下：

```
//数组元素逆序
public static void receive(int[] arr){
    for (int start = 0, end = arr.length-1; start < end; start++,end--) {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
    }
}
```

2.3 数组元素选择排序

图解：



题目分析：

通过观察发现，本题目要实现把数组元素{13,46,22,65,3}进行排序

1. 提到数组排序，就要进行元素值大小的比较，通过上图发现，我们想完成排序要经过若干次的比较才能够完成。
2. 上图中用每圈要比较的第一个元素与该元素后面的数组元素依次比较到数组的最后一个元素，把小的值放在第一个数组元素中，数组循环一圈后，则把最小元素值互换到了第一个

元素中。

3. 数组再循环一圈后，把第二小的元素值互换到了第二个元素中。按照这种方式，数组循环多圈以后，就完成了数组元素的排序。这种排序方式我们称为选择排序。

解题步骤：

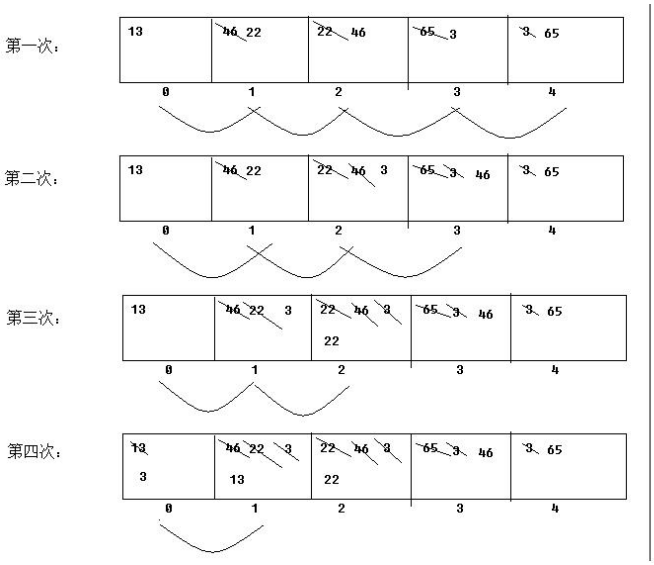
1. 使用 for 循环（外层循环），指定数组要循环的圈数（通过图解可知，数组循环的圈数为数组长度 - 1）
2. 在每一圈中，通过 for 循环（内层循环）完成数组要比较的第一个元素与该元素后面的数组元素依次比较到数组的最后一个元素，把小的值放在第一个数组元素中
3. 在每一圈中，要参与比较的第一个元素由第几圈循环来决定。如上图所示
   a) 进行第一圈元素比较时，要比较的第一个元素为数组第一个元素，即索引为 0 的元素
   b) 进行第二圈元素比较时，要比较的第一个元素为数组第二个元素，即索引为 1 的元素
   c) 依次类推，得出结论：进行第 n 圈元素比较时，要比较的第一个元素为数组第 n 个元素，即数组索引为 n-1 的元素

代码如下：

```
//选择排序
public static void selectSort(int[] arr) {
    //功能
    //外层循环用来控制数组循环的圈数
    for (int i = 0; i < arr.length-1; i++) {
        //内层循环用来完成元素值比较，把小的元素值互换到要比较的第一个元素中
        for (int j = i+1; j < arr.length; j++) {
            if (arr[i] > arr[j]) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}
```

2.4 数组元素冒泡排序

图解：数组元素{13,46,22,65,3}




```
0  第一次：
        arr[0] 与 arr[1]      j < 5-1 -0
        arr[1] 与 arr[2]
        arr[2] 与 arr[3]
        arr[3] 与 arr[4]
1  第二次：                   j < 5-1 -1
        arr[0] 与 arr[1]
        arr[1] 与 arr[2]
        arr[2] 与 arr[3]
2  第三次：                   j < 5-1 -2
        arr[0] 与 arr[1]
        arr[1] 与 arr[2]
3  第四次：                   j < 5-1 -3
        arr[0] 与 arr[1]

                              j < arr.length-1 - i;
外层循环用来控制比较的次数
内层循环用来控制参与比较的元素
```

题目分析：

通过观察发现，本题目要实现把数组元素{13,46,22,65,3}进行排序

1. 提到数组排序，就要进行元素值大小的比较，通过上图发现，我们想完成排序要经过若干次的比较才能够完成。
2. 上图中相邻的元素值依次比较，把大的值放后面的元素中，数组循环一圈后，则把最大元素值互换到了最后一个元素中。数组再循环一圈后，把第二大的元素值互换到了倒数第二个元素中。按照这种方式，数组循环多圈以后，就完成了数组元素的排序。这种排序方式我们称为冒泡排序。

解题步骤：

1. 使用 for 循环（外层循环），指定数组要循环的圈数（通过图解可知，数组循环的圈数为数组长度 - 1）
2. 在每一圈中，通过 for 循环（内层循环）完成相邻的元素值依次比较，把大的值放后面的元素中

3. 每圈内层循环的次数，由第几圈循环来决定。如上图所示

   a) 进行第一圈元素比较时，内层循环次数为数组长度 - 1

   b) 进行第二圈元素比较时，内层循环次数为数组长度 - 2

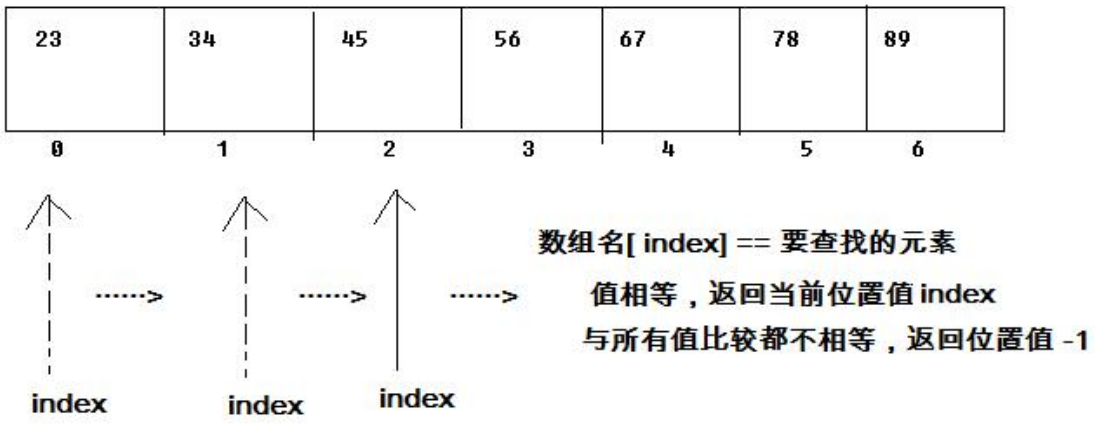
   c) 依次类推，得出结论：进行第 n 圈元素比较时，内层循环次数为数组长度 - n

代码如下：

```
//冒泡排序
public static void bubbleSort(int[] arr) {
    //功能
    //外层循环用来控制数组循环的圈数
    for (int i = 0; i < arr.length-1; i++) {
        //j < arr.length-1  为了避免角标越界
        //j < arr.length-1-i  为了比较效率,避免重复比较
        //内层循环用来完成元素值比较，把大的元素值互换到后面
        for (int j = 0; j < arr.length-1-i; j++) {
            if (arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
```

2.5 数组元素普通查找

图解：



题目分析：

通过观察发现 ,本题目要实现查找指定数值第一次在数组中存储的位置(索引),返回该位置(索引)。

1. 我们可以通过遍历数组，得到每个数组元素的值
2. 在遍历数组过程中，使用当前数组元素值与要查找的数值进行对比
   a) 数值相等，返回当前数组元素值的索引
   b) 整个循环结束后，比对结果数值没有相等的情况，说明该数组中没有存储要查找的数值，此时，返回一个索引值-1，来表示没有查询到对应的位置。(使用 -1 来表示没有查询到，是因为数组的索引没有负数)

解题步骤：

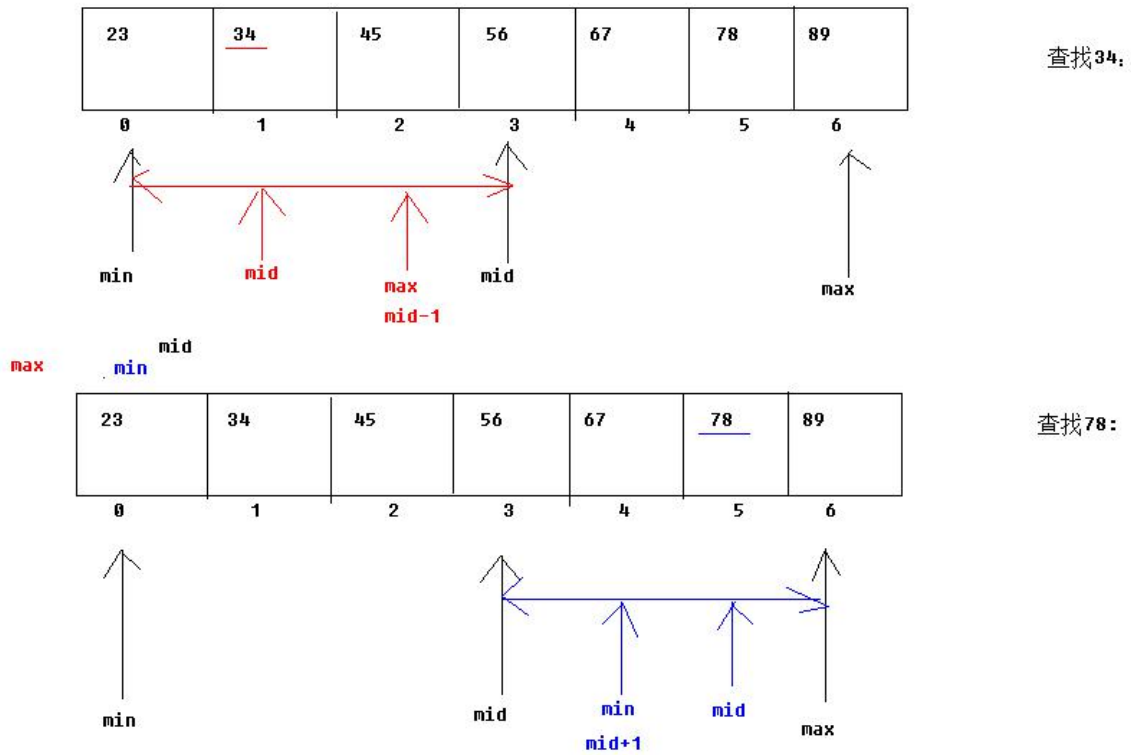
1. 使用 for 循环，遍历数组，得到每个数组元素值
2. 在每次循环中，使用 if 条件语句进行当前数组元素值与要查找的数值进行对比，若比较结果相等，直接返回当前数组元素的索引值
3. 若整个循环结束后，比对结果数值没有相等的情况，说明该数组中没有存储要查找的数值，此时，返回一个索引值-1

代码如下：

```
//普通查找
public static int getArrayIndex(int[] arr, int number) {
    //把数组中的元素依次与指定的数值 进行比较
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == number) {
            //找到了
            return i;
        }
    }
    return -1;
}
```

2.6 数组元素二分查找（折半查找）

图解：



题目分析：

通过观察发现，本题目要实现查找指定数值在**元素有序的数组**中存储的位置（索引），返回该位置（索引）。

1. 我们使用数组最中间位置的元素值与要查找的指定数值进行比较，若相等，返回中间元素值的索引
2. 最中间位置的元素值与要查找的指定数值进行比较，若不相等，则根据比较的结果，缩小查询范围为上次数组查询范围的一半；

   再根据新的查询范围，更新最中间元素位置，然后使用中间元素值与要查找的指定数值进行比较

   - 比较结果相等，返回中间元素值的索引
   - 比较结果不相等，继续缩小查询范围为上次数组查询范围的一半，更新最中间元素位置，继续比较，依次类推。

3. 当查询范围缩小到小于 0 个元素时，则指定数值没有查询到，返回索引值-1。

解题步骤：

1. 定义 3 个用来记录索引值的变量，变量 min 记录当前范围最小索引值，初始值为 0；变量 max 记录当前范围最大索引值，初始值为数组长度-1；变量 mid 记录当前当前范围最中间元素的索引值，初始值为(min+max) / 2
2. 使用循环，判断当前范围下，最中间元素值与指定查找的数值是否相等
   - 若相等，结束循环，返回当前范围最中间元素的索引值 mid
   - 若不相等，根据比较结果，缩小查询范围为上一次查询范围的一般
     - 中间元素值 比 要查询的数值大，说明要查询的数值在当前范围的最小索引位置与中间索引位置之间，此时，更新查询范围为:

```
范围最大索引值 = 上一次中间索引位置 -1；
```

     - 中间元素值 比 要查询的数值小，说明要查询的数值在当前范围的最大索引位置与中间索引位置之间，此时，更新查询范围为:

```
范围最小索引值 = 上一次中间索引位置 +1；
```

- 在新的查询范围中，更新中间元素值的位置，再次使用最中间元素值与指定查找的数值是否相等。

```
中间索引值 = (范围最小索引值 +范围最大索引值) / 2;
```

3. 每次查询范围缩小一半后，使用 if 语句判断，查询范围是否小于 0 个元素，若小于 0 个元素，则说明指定数值没有查询到，返回索引值-1。

代码如下：

```
//二分查找法(折半查找法)
public static int halfSearch(int[] arr, int number) {
    //定义 3 个变量，用来记录 min, min, mid 的位置
    int min = 0;
    int max = arr.length-1;
    int mid = 0;
        while (min <= max) {
          mid = (min+max)/2;
        //没找了， 更新范围，继续比较
        //更新范围
        if (arr[mid] > number) {
            //在左边
            max = mid-1;
        } else if(arr[i] < number){
            //在右边
            min = mid+1;
        }
        else{
            return mid ;
         }

    return -1;
}
```