数组变量

直接初始化数组

直接初始化数组

- new创建的数组会得到默认的0值

直接初始化数组

- new创建的数组会得到默认的0值
- int[] scores = {87, 98, 69, 54, 65, 76, 87, 99};

直接初始化数组

- new创建的数组会得到默认的0值
- int[] scores = {87, 98, 69, 54, 65, 76, 87, 99};
- 直接用大括号给出数组的所有元素的初始值

直接初始化数组

- new创建的数组会得到默认的0值
- int[] scores = {87, 98, 69, 54, 65, 76, 87, 99};
- 直接用大括号给出数组的所有元素的初始值
- 不需要给出数组的大小，编译器替你数数

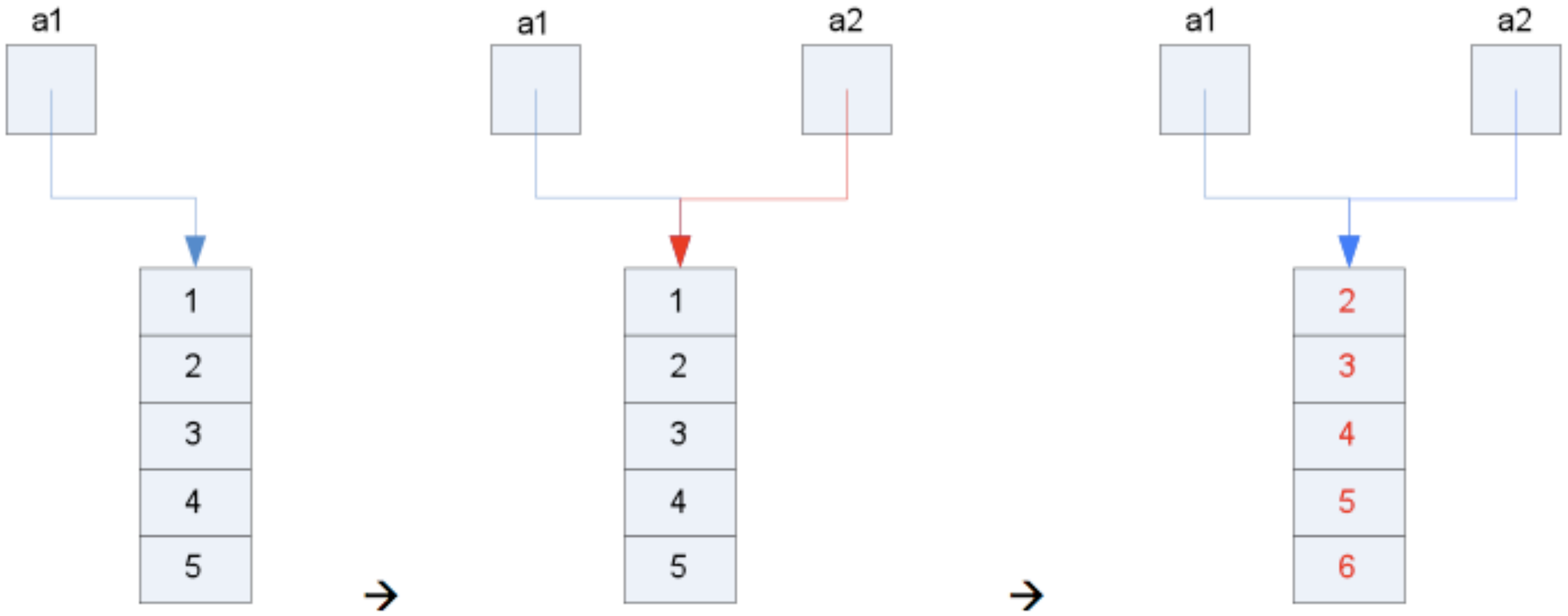
直接初始化数组

- new创建的数组会得到默认的0值
- int[] scores = {87, 98, 69, 54, 65, 76, 87, 99};
- 直接用大括号给出数组的所有元素的初始值
- 不需要给出数组的大小，编译器替你数数
- 如何得知数组的大小？length！

数组变量赋值

```
int[] a = new int[10];

a[0] = 5;

int[] b = a;

b[0] = 16;

System.out.println(a[0]);
```

```
int[] a1 = {1,2,3,4,5};
int[] a2 = a1;
for ( int i=0; i<a2.length; ++i )
{
  a2[i] ++;
}
for ( int i=0; i<a1.length; ++i )
{
  System.out.println(a1[i]);
}
```


a1
1
2
3
4
5
a1
a2
1
2
3
4
5
a1
a2
2
3
4
5
6

数组变量

- 数组变量是数组的管理者而非数组本身
- 数组必须创建出来然后交给数组变量来管理
- 数组变量之间的赋值是管理权限的赋予
- 数组变量之间的比较是判断是否管理同一个数组

复制数组

- 必须遍历源数组将每个元素逐一拷贝给目的数组

遍历数组

搜索

- 在一组给定的数据中，如何找出某个数据是否存在?

```
int[] data = {2, 3, 5, 7, 4, 9, 11, 34, 12, 28};
int x = in.nextInt();
int loc = -1;
for ( int i=0; i<data.length; i++ )
{
  if ( x == data[i] )
  {
    loc = i;
    break;
  }
}
if ( loc > -1 )
{
  System.out.println(x+"是第"+(loc+1)+"个");
}
else
{
  System.out.println(x+"不在其中");
}
```

遍历数组

```
for ( int i=0; i<data.length; i++ )
{
  if ( x == data[i] )
  {
    loc = i;
    break;
  }
}
```

遍历数组

```
for ( int i=0; i<data.length; i++ )
{
  if ( x == data[i] )
  {
    loc = i;
    break;
  }
}
```

- 通常都是使用for循环，让循环变量i从0到<数组的length，这样循环体内最大的i正好是数组最大的有效下标

遍历数组

```
for ( int i=0; i<data.length; i++ )
{
  if ( x == data[i] )
  {
    loc = i;
    break;
  }
}
```

- 通常都是使用for循环，让循环变量i从0到<数组的length，这样循环体内最大的i正好是数组最大的有效下标
- 常见错误是：

遍历数组

```
for ( int i=0; i<data.length; i++ )
{
  if ( x == data[i] )
  {
    loc = i;
    break;
  }
}
```

- 通常都是使用for循环，让循环变量i从0到<数组的length，这样循环体内最大的i正好是数组最大的有效下标
- 常见错误是：
  - 循环结束条件是<=数组长度，或；

遍历数组

```
for ( int i=0; i<data.length; i++ )
{
  if ( x == data[i] )
  {
    loc = i;
    break;
  }
}
```

- 通常都是使用for循环，让循环变量i从0到<数组的length，这样循环体内最大的i正好是数组最大的有效下标
- 常见错误是：
  - 循环结束条件是<=数组长度，或；
  - 离开循环后，继续用i的值来做数组元素的下标！

```java
int[] data = {2, 3, 5, 7, 4, 9, 11, 34, 12, 28};
int x = in.nextInt();
int loc = -1;
for ( int i=0; i<data.length; i++ )
{
  if ( x == data[i] )
  {
    loc = i;
    break;
  }
}
if ( loc > -1 )
{
  System.out.println(x+"是第"+(loc+1)+"个");
}
else
{
  System.out.println(x+"不在其中");
}
```

```java
int[] data = {2, 3, 5, 7, 4, 9, 11, 34,
int x = in.nextInt();
boolean found = false;
for ( int k : data )
{
  if ( x == k )
  {
    found = true;
    break;
  }
}
if ( found )
{
  System.out.println(x+"在其中");
}
else
{
  System.out.println(x+"不在其中");
}
```

for-each

```
for ( <类型> <变量>: <数组> ) {
    ...
}
```

数组的例子：素数

```
int x = in.nextInt();
boolean isPrime = true;
if ( x == 1 )
{
  isPrime = false;
}
for ( int i=2; i<x; i++ )
{
  if ( x % i == 0 )
  {
    isPrime = false;
    break;
  }
}
if ( isPrime )
{
  System.out.println(x+"是素数");
}
else
{
  System.out.println(x+"不是素数");
}
```

从2到x-1测试是否可以整除

- 对于n要循环n-1遍
- 当n很大时就可以被看作是n遍

去掉偶数后，从3到x-1，每次加2

```
if ( x == 1 || x%2 == 0 && x!= 2 )
{
  isPrime = false;
}
else
{
  for ( int i=3; i<x; i+=2 )
  {
    if ( x % i == 0 )
    {
      isPrime = false;
      break;
    }
  }
}
```

- 如果x是偶数，立刻
- 否则要循环(n-3)/2+1遍
  - 当n很大时就是n/2遍

无须到x-1，到sqrt(x)就够了

```
for ( int i=3; i<Math.sqrt(x); i+=2 )
{
  if ( x % i == 0 )
  {
    isPrime = false;
    break;
  }
}
```

- 只需要循环sqrt(x)遍
- 从n—>n/2—>sqrt(n)

判断是否能被已知的且<x的素数整除

- 构造前50个素数的表

判断是否能被已知的且<x的素数整除

- 构造前50个素数的表

```
int[] primes = new int[50];
primes[0] = 2;
int cnt = 1;
MAIN_LOOP:
for ( int x = 3; cnt<primes.length; x++ )
{
  for ( int i=0; i<cnt; i++ )
  {
    if ( x % primes[i] == 0 )
    {
      continue MAIN_LOOP;
    }
  }
  primes[cnt++] = x;
}
for ( int k : primes )
{
  System.out.print(k+" ");
}
System.out.println();
```

构造素数表

- 欲构造n以内的素数表

1. 令x为2
2. 将2x、3x、4x直至ax<n的数标记为非素数
3. 令x为下一个没有被标记为非素数的数，重复2；直到所有的数都已经尝试完毕

构造素数表

- 欲构造n以内（不含）的素数表

1. 创建prime为boolean[n]，初始化其所有元素为true，prime[x]为true表示x是素数
2. 令x=2
3. 如果x是素数，则对于(i=2;x*i<n;i++)令prime[i*x]=false
4. 令x++，如果x<n，重复3，否则结束

构造素数表

```
boolean[] isPrime = new boolean[100];
for ( int i=2; i<isPrime.length; i++ )
{
   isPrime[i] = true;
}

for ( int i=2; i<isPrime.length; i++ )
{
   if ( isPrime[i] )
   {
      for ( int k=2; i*k<isPrime.length; k++ )
      {
         isPrime[i*k] = false;
      }
   }
}
for ( int i = 0; i<isPrime.length; i++ )
{
   if ( isPrime[i] )
   {
      System.out.print(i+" ");
   }
}
System.out.println();
```

- 算法不一定和人的思考方式相同

二维数组

二维数组

- int[][] a = new int[3][5];
- 通常理解为a是一个3行5列的矩阵

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]

二维数组的遍历

```
for ( i=0; i<3; i++ ) {
    for ( j=0; j<5; j++ ) {
        a[i][j] = i*j;
    }
}
```

- a[i][j]是一个int
- 表示第i行第j列上的单元
  - a[i,j]并不存在

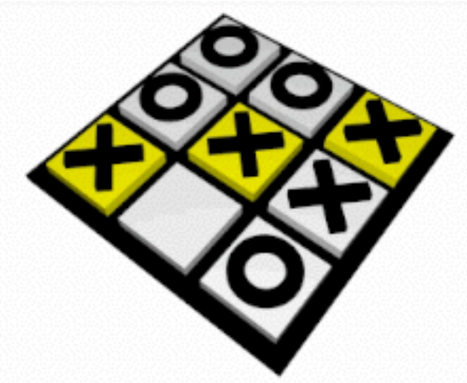
二维数组的初始化

```
int[][] a = {
    {1,2,3,4},
    {1,2,3},
};
```

- 编译器来数数
- 每行一个{}，逗号分隔
- 最后的逗号可以存在，有古老的传统
- 如果省略，表示补零

tic-tac-toe游戏

- 读入一个3X3的矩阵，矩阵中的数字为1表示该位置上有一个X，为0表示为O
- 程序判断这个矩阵中是否有获胜的一方，输出表示获胜一方的字符X或O，或输出无人获胜



本页图片来源：wikipedia.org

读入矩阵

```
final int SIZE = 3;
int[][] board = new int[SIZE][SIZE];
boolean gotResult = false;
int numOfX = 0;
int numOfO = 0;

// 读入矩阵
for ( int i=0; i<SIZE; i++ )
{
  for ( int j=0; j<SIZE; j++ )
  {
    board[i][j] = in.nextInt();
  }
}
```

检查行

```
// 检查行
for ( int i=0; i<SIZE; i++ )
{
  numOfX = 0;
  numOfO = 0;
  for ( int j=0; i<SIZE; i++ )
  {
    if ( board[i][j] == 1 )
    {
      numOfX ++;
    }
    else
    {
      numOfO ++;
    }
  }
  if ( numOfX == SIZE || numOfO == SIZE )
  {
    gotResult = true;
    break;
  }
}
```

检查列

```
//  检查列
if ( !gotResult )
{
  for ( int i=0; i<SIZE; i++ )
  {
    numOfX = 0;
    numOfO = 0;
    for ( int j=0; i<SIZE; i++ )
    {
      if ( board[j][i] == 1 )
      {
        numOfX ++;
      }
      else
      {
        numOfO ++;
      }
    }
    if ( numOfX == SIZE || numOfO == SIZE )
    {
      gotResult = true;
      break;
    }
  }
}
```

行和列?

```
// 检查行
for ( int i=0; i<SIZE; i++ )
{
   numOfX = 0;
   numOfO = 0;
   for ( int j=0; i<SIZE; i++ )
   {
      if ( board[i][j] == 1 )
      {
         numOfX ++;
      }
      else
      {
         numOfO ++;
      }
   }
   if ( numOfX == SIZE || numOfO == SIZE )
   {
      gotResult = true;
      break;
   }
}
```

```
// 检查列
if ( !gotResult )
{
   for ( int i=0; i<SIZE; i++ )
   {
      numOfX = 0;
      numOfO = 0;
      for ( int j=0; i<SIZE; i++ )
      {
         if ( board[j][i] == 1 )
         {
            numOfX ++;
         }
         else
         {
            numOfO ++;
         }
      }
      if ( numOfX == SIZE || numOfO == SIZE )
      {
         gotResult = true;
         break;
      }
   }
}
```

行和列?

```
// 检查行
for ( int i=0; i<SIZE; i++ )
{
   numOfX = 0;
   numOfO = 0;
   for ( int j=0; i<SIZE; i++ )
   {
      if ( board[i][j] == 1 )
      {
         numOfX ++;
      }
      else
      {
         numOfO ++;
      }
   }
   if ( numOfX == SIZE || numOfO == SIZE )
   {
      gotResult = true;
      break;
   }
}
```

能否用一个两重循环来检查行和列?

```
// 检查列
if ( !gotResult )
{
   for ( int i=0; i<SIZE; i++ )
   {
      numOfX = 0;
      numOfO = 0;
      for ( int j=0; i<SIZE; i++ )
      {
         if ( board[j][i] == 1 )
         {
            numOfX ++;
         }
         else
         {
            numOfO ++;
         }
      }
      if ( numOfX == SIZE || numOfO == SIZE )
      {
         gotResult = true;
         break;
      }
   }
}
```

检查对角线

```
// 检查对角线
if ( !gotResult )
{
  numOfX = 0;
  numOfO = 0;
  for ( int i=0; i<SIZE; i++ )
  {
    if ( board[i][i] == 1 )
    {
      numOfX ++;
    }
    else
    {
      numOfO ++;
    }
  }
  if ( numOfX == SIZE || numOfO == SIZE )
  {
    gotResult = true;
  }
}
```

```
// 检查反对角线
if ( !gotResult )
{
  numOfX = 0;
  numOfO = 0;
  for ( int i=0; i<SIZE; i++ )
  {
    if ( board[i][SIZE-i-1] == 1 )
    {
      numOfX ++;
    }
    else
    {
      numOfO ++;
    }
  }
  if ( numOfX == SIZE || numOfO == SIZE )
  {
    gotResult = true;
  }
}
```

输出结果

```
if ( gotResult )
{
  if ( numOfX == SIZE )
  {
    System.out.println("X WIN");
  }
  else
  {
    System.out.println("O WIN");
  }
}
```