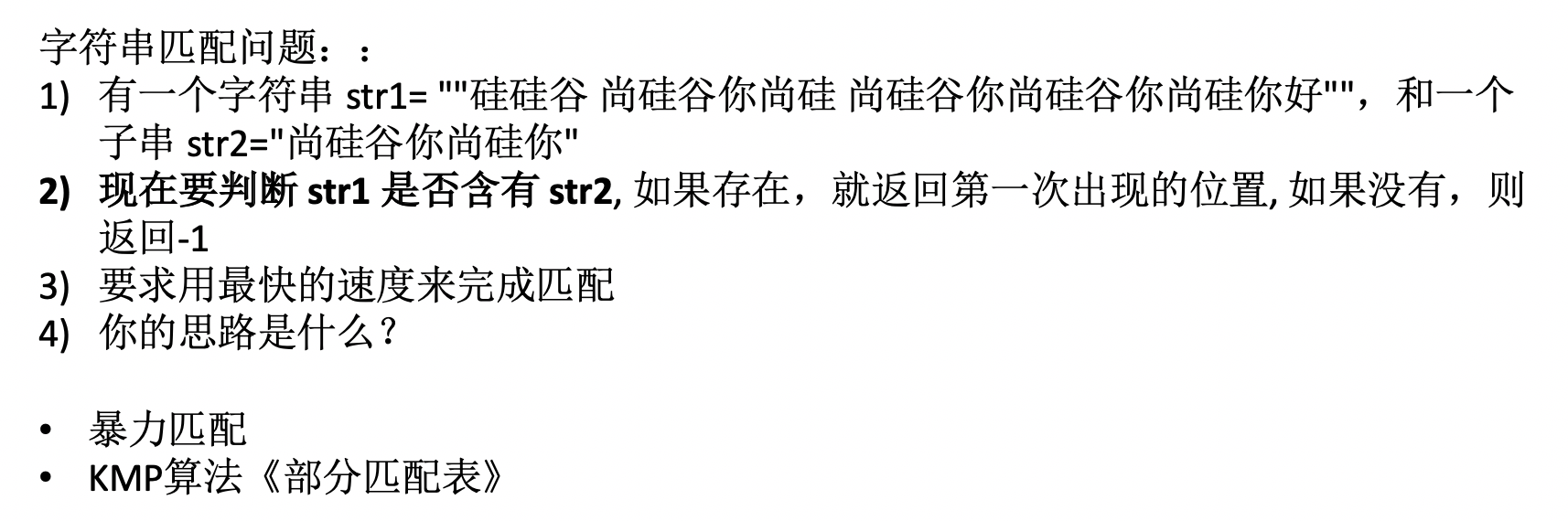
# 数据结构和算法

## 1、尚硅谷-Java数据结构和算法

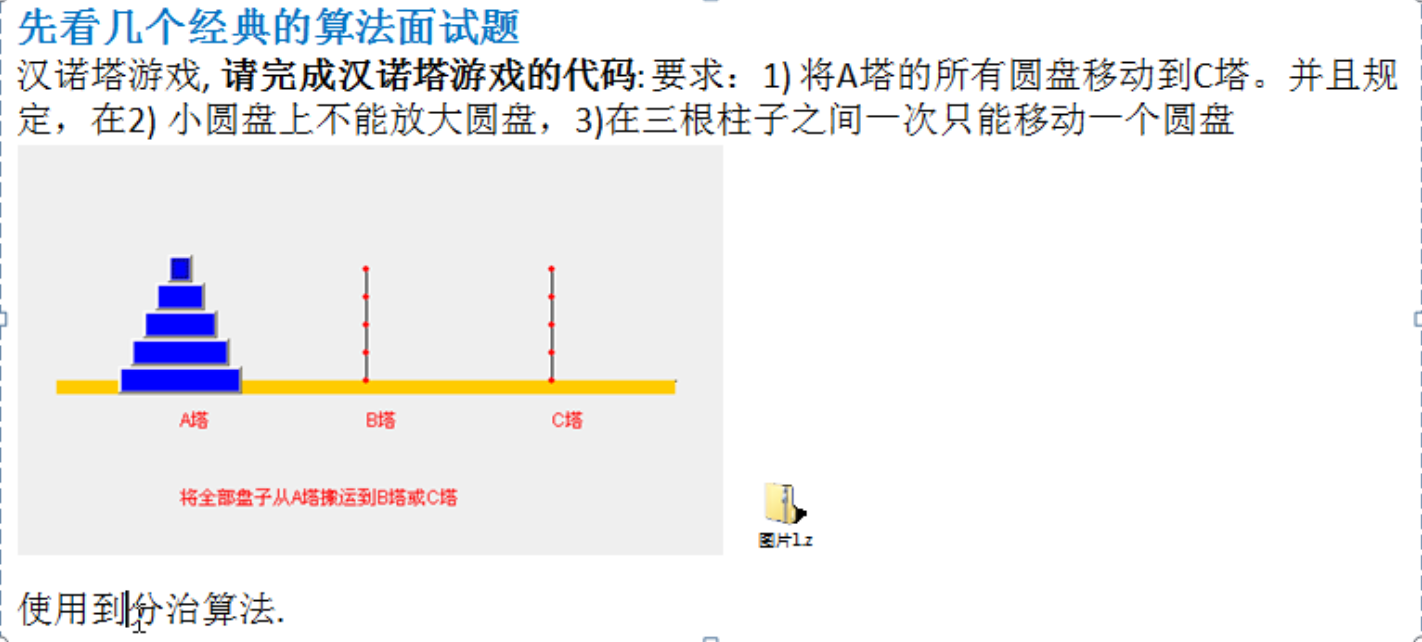
### 1． 几个经典的算法面试题

#### 1.1 字符串匹配算法：（KMP算法）



暴力匹配：简单，但是效率低

#### 1.2 汉诺塔问题（分治算法）



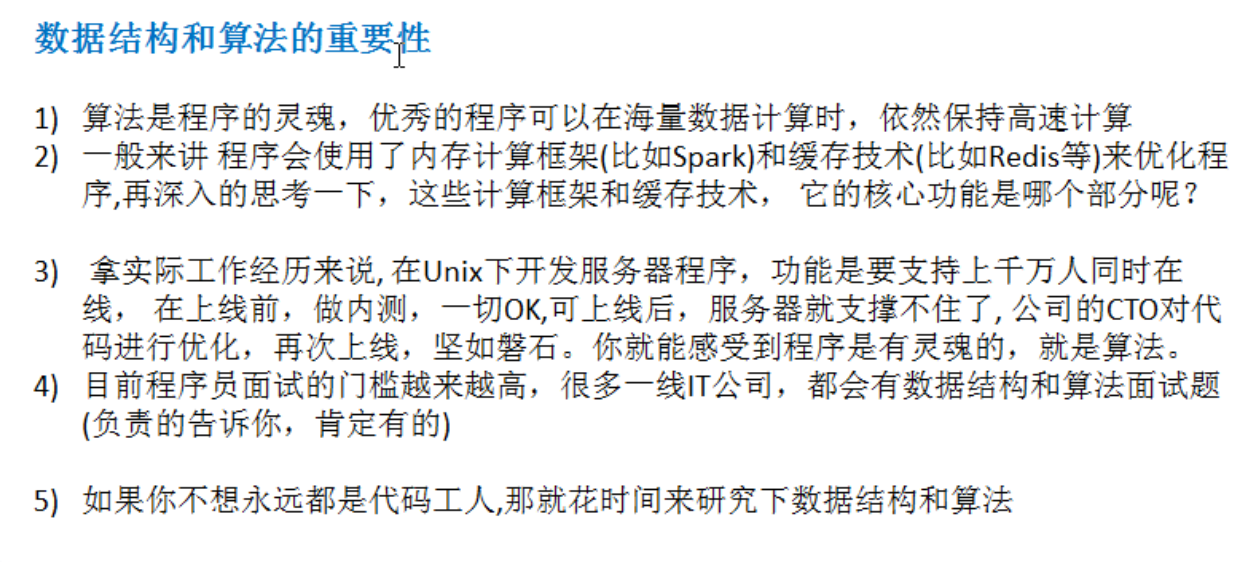
#### 1.3 八皇后问题：（回溯算法）



#### 1.4 马踏棋盘算法：

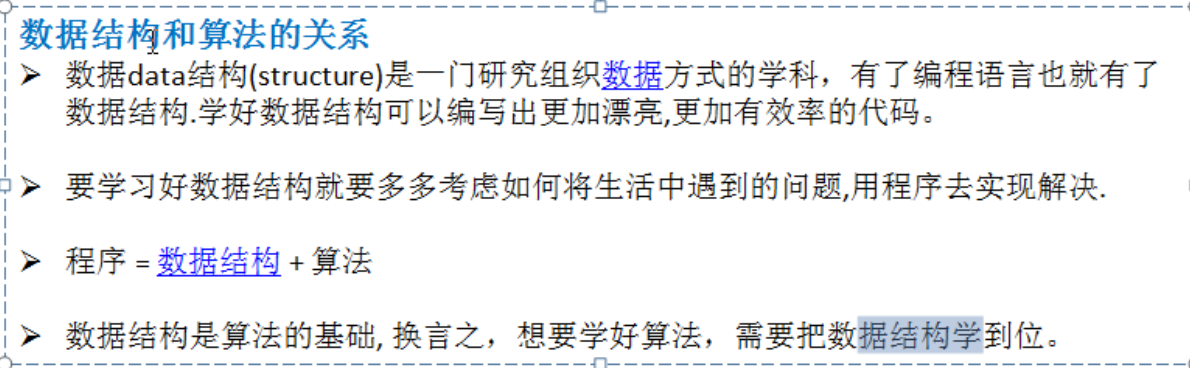


### 2 数据结构和算法的重要性



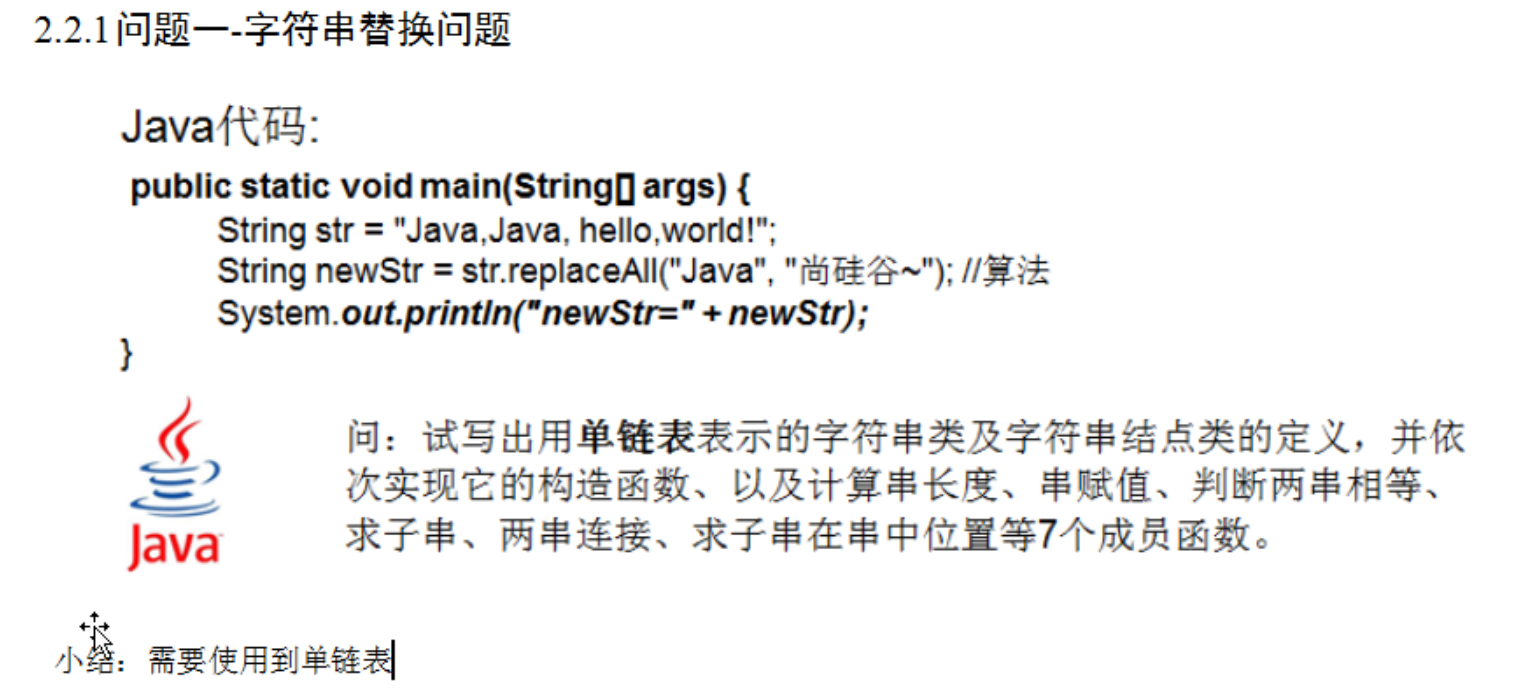
### 3．数据结构和算法介绍

#### 3.1数据结构和算法的关系

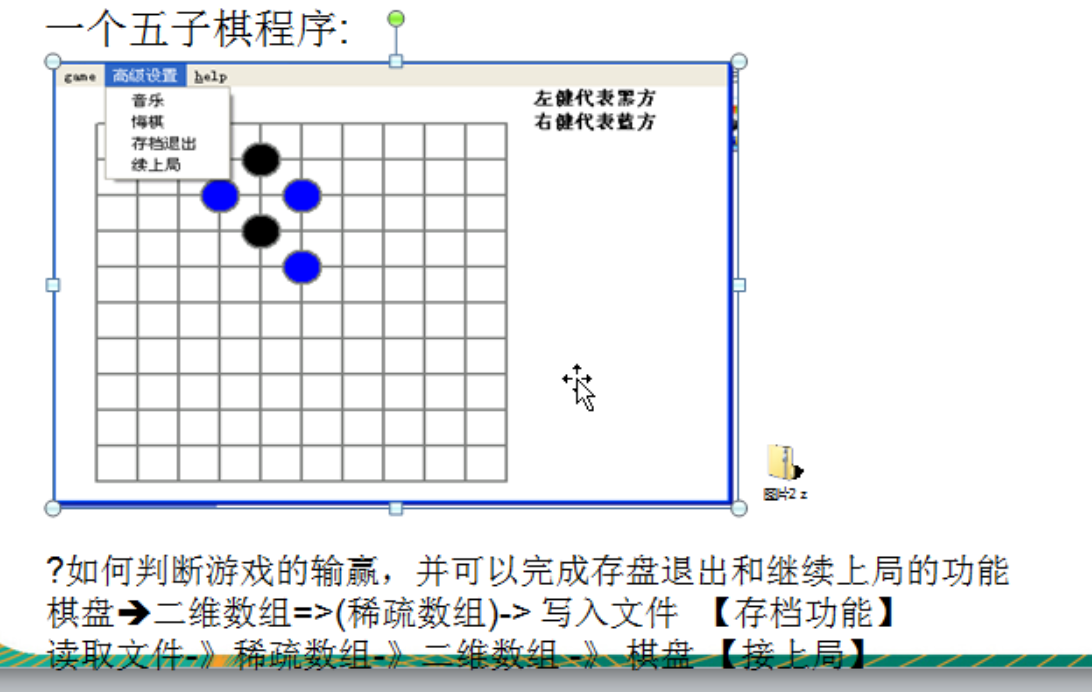


#### 3.2 编程中遇到的问题：

##### 3.2.1 字符串替换问题：



##### 3.2.2 一个五子棋程序：



1. 棋盘->二维数组=>（稀疏数组）->写入文件【存档功能】
2. 读取文件-》稀疏数组-》二维数组-》棋盘【接上局】

##### 3.2.3 约瑟夫问题(丢手帕问题)：



3）小结：完成约瑟夫问题，需要使用到单向环形链表这个数据结构

##### 3.2.4 其它常见问题：



1. 修路问题=>最小生成树(加权值)【数据结构】+普利姆算法
2. 最短路径问题=>图+弗洛尹德算法
3. 汉诺塔=>分支算法
4. 八皇后问题=>回溯法

#### 3.3 线性结构和非线性结构

数据结构包括：线性结构和非线性结构。

##### 3.3.1 线性结构

1. 线性结构作为最常用的数据结构，其特点是**数据元素之间存在一对一**的线性关系。
2. 线性结构有两种不同的存储结构，即**顺序存储结构（数组）**和**链式存储结构（链表）**。顺序存储的线性表成为顺序表，顺序表中的**存储元素是连续**（即存储地址是连续）的。
3. 链式存储的线性表成为链表，链表中的**存储元素不一定是连续的**，元素节点中存放数据元素以及相邻的元素的地址信息。
4. 线性结构常见的有：**数组、队列、链表和栈**，后面详细讲解。

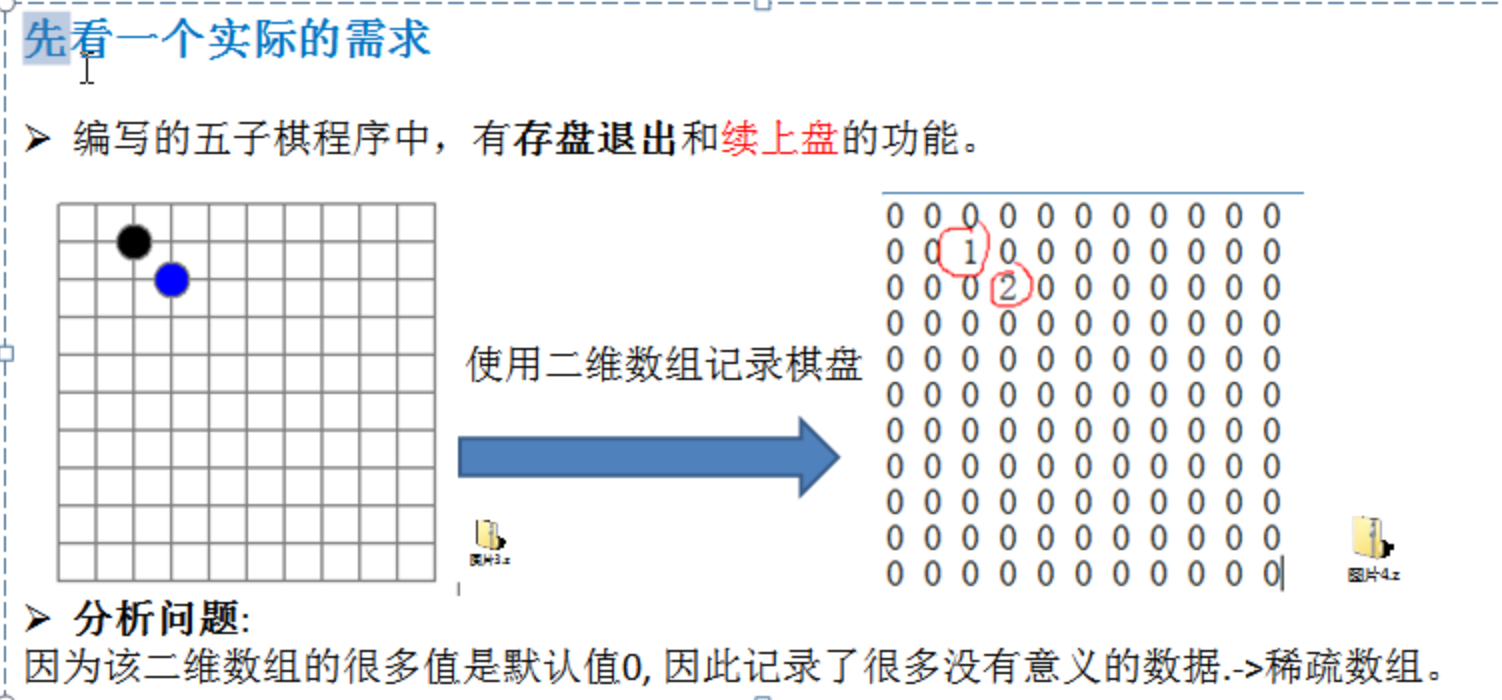
##### 3.3.2 非线性结构

非线性结构包括：二维数组、多维数组，广义表、树结构、图结构。

### 4、 稀疏数组和队列

#### 4.1 稀疏sparsearray数组

##### 4.1.1 一个实际需求



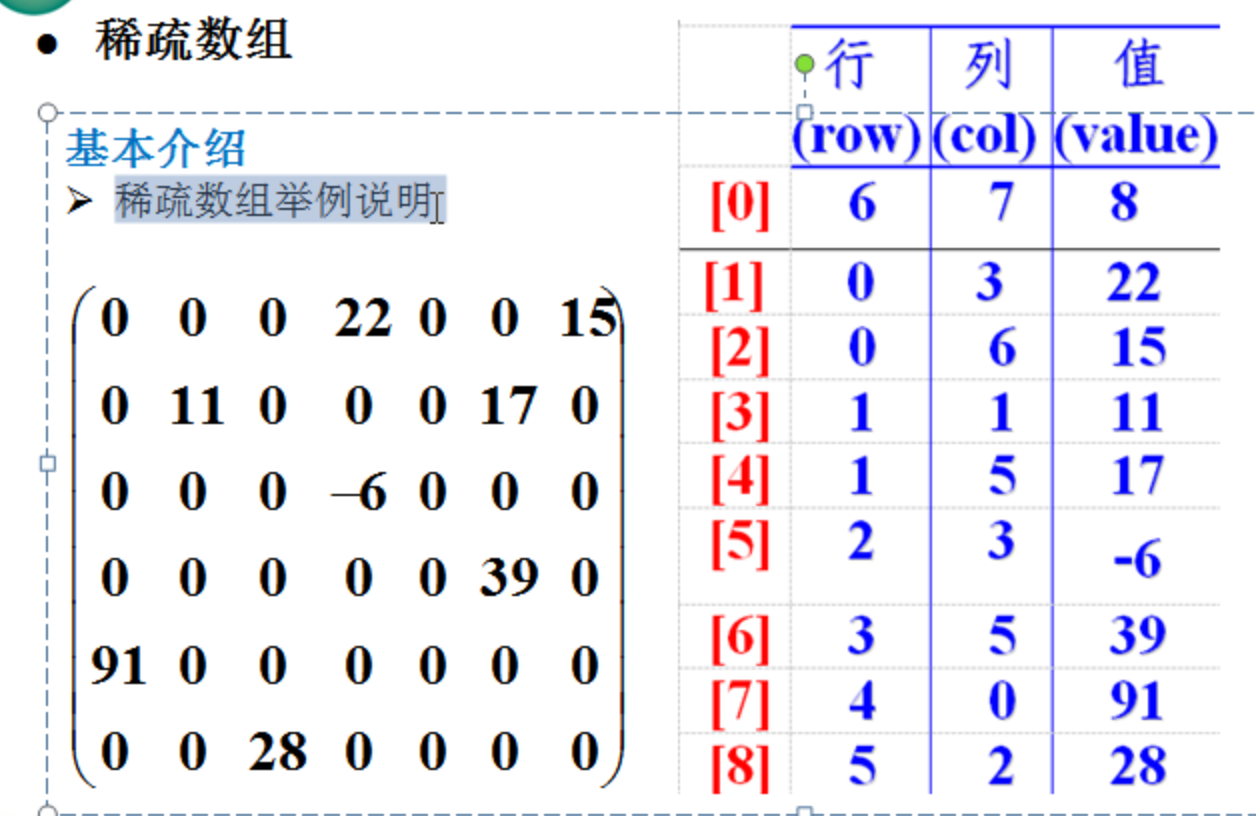
##### 4.1.2 基本介绍

当一个数组中大部分元素为0，或者为同一个值得数组时，可以使用稀疏数组来保存该数组。

稀疏数组的处理方法是：

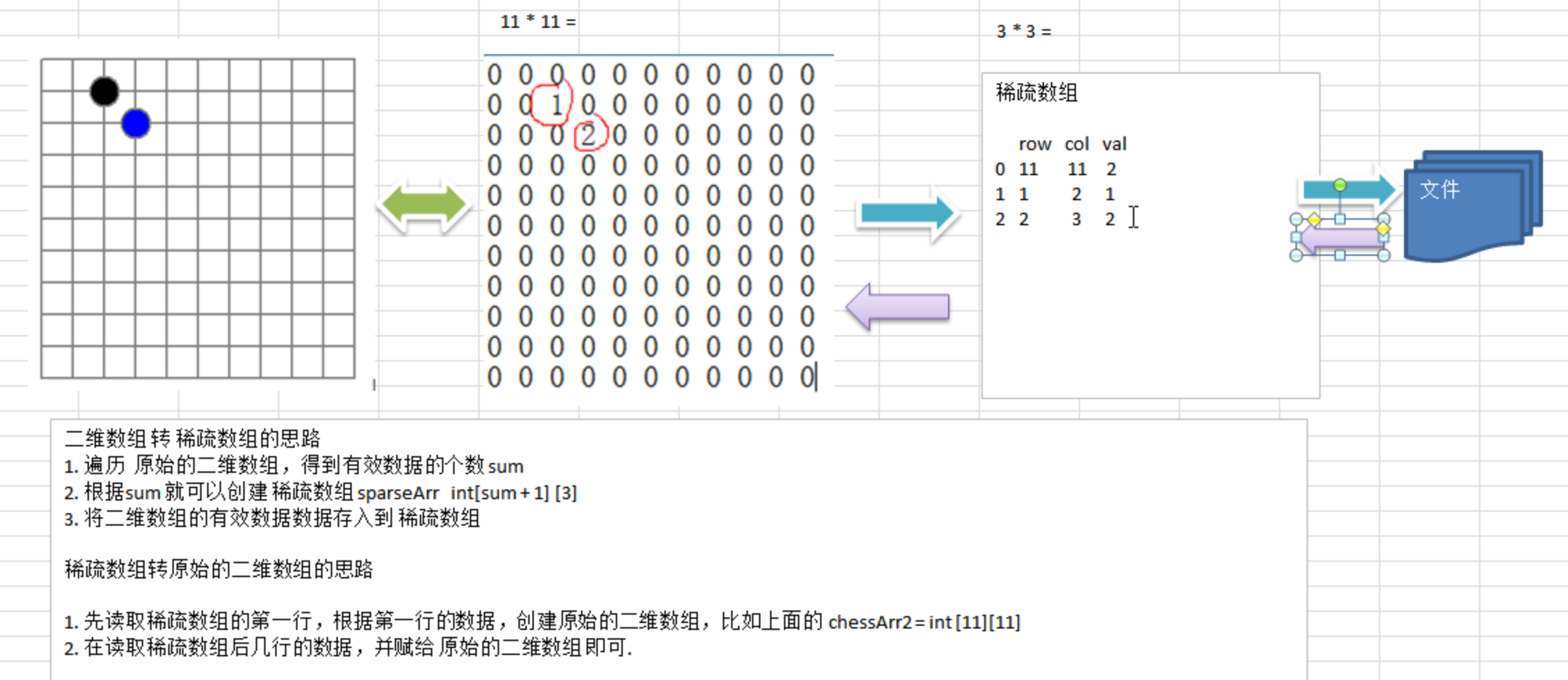
* + 1. 记录数组一共有几行几列，有多少个不同的值
    2. 把具有不同值得元素的行列及值记录在一个小规模的数组中，从而缩小程序的规模

###### 稀疏数组举例说明：



##### 4.1.3 应用实例

1. 使用稀疏数组，来保留类似前面的二维数组（棋盘、地图等等）
2. 把稀疏数组存盘，并且可以重新恢复原来的二维数组
3. 整理思路分析



##### 4.1.4 代码实现

|  |
| --- |
| **package** com.atguigu.sparsearray;  **public class** SparseArray {  **public static void** main(String[] args) {  *//创建一个原始的二维数组 11 \* 11  //0:表示没有棋子，1：表示黑子 2 表篮子* **int** chessArr1[][] = **new int**[11][11];  chessArr1[1][2] = 1;  chessArr1[2][3] = 2;  chessArr1[3][5] = 2;  System.***out***.println(**"原始的二维数组："**);  **for**(**int**[] row : chessArr1){  **for** (**int** data : row){  System.***out***.printf(**"%d\t"**,data);  }  System.***out***.println();  }   *//1、将原始的二维数组 转稀疏数组  //1. 先遍历二维数组，得到非0数据的个数* **int** sum = 0;  **for**(**int** i=0;i<chessArr1.**length**;i++){  **for**(**int** j=0;j<chessArr1[i].**length**;j++){  **if**(chessArr1[i][j] != 0){  sum++;  }  }  }  System.***out***.println(**"sum="** + sum);   *//2.创建对应的稀疏数组* **int** sparseArray[][] = **new int**[sum+1][3];  *//给稀疏数组赋值* sparseArray[0][0] = chessArr1.**length**;  sparseArray[0][1] = chessArr1[0].**length**;  sparseArray[0][2] = sum;   *//遍历二维数组，将非0的值存放到sparseArray* **int** k=0;*//用于记录是第几个非零数据* **for**(**int** i=0;i<chessArr1.**length**;i++){  **for**(**int** j=0;j<chessArr1[i].**length**;j++){  **if**(chessArr1[i][j] != 0){  k++;  sparseArray[k][0] = i;  sparseArray[k][1] = j;  sparseArray[k][2] = chessArr1[i][j];   }  }  }  *//输出稀疏数组的形式* System.***out***.println();  System.***out***.println(**"得到的稀疏数组为："**);  **for**(**int** i=0;i<sparseArray.**length**;i++){  System.***out***.printf(**"%d\t%d\t%d\t\n"**,sparseArray[i][0],sparseArray[i][1],sparseArray[i][2]);  }  System.***out***.println();    *//将稀疏素组 --》 恢复成原始的二维数组  /\*  1.先读取稀疏数组的第一行，根据第一行的数据，创建原始的二维数组，比如上面的 chessArr2 = int[11][11];  2.在读取稀疏数组后几行的数据，并赋给原始的二维数组即可  \*/   //1.先读取稀疏数组的第一行，根据第一行的数据，创建原始的二维数组* **int** chessArr2[][] = **new int**[sparseArray[0][0]][sparseArray[0][1]];   *//2.在读取稀疏数组后几行的数据(从第二行开始)，并赋给原始的二维数组即可* **for** (**int** i = 1;i < sparseArray.**length**;i++){  chessArr2[sparseArray[i][0]][sparseArray[i][1]] = sparseArray[i][2];  }  *//输出恢复后的二维数组* System.***out***.println();  System.***out***.println(**"恢复后的二维数组："**);  **for**(**int**[] row : chessArr2){  **for** (**int** data : row){  System.***out***.printf(**"%d\t"**,data);  }  System.***out***.println();  }    } } |