* 今天学习目标

  * 能够理解数据结构的概述

    * 存储，组织数据一种方式

    * 优雅，高效

  * 能够理解数据结构之栈

    * Stack：FILO

      * Stack怎么样去使用（push，pop）

      * 递归（方法调用压栈的过程，方法返回（return）弹栈过程）

  * 能够理解数据结构之队列

    * Queue：FIFO

    * LinkedList

  * 能够理解数据结构之数组

    * 为什么增删比较慢

    * 稀疏数组

  * 能够理解数据结构之链表

    * 自习这个链表（笔记+百度文章）

    * 为什么查询慢，增删快？？？

  * 能够实现自定义LinkedList

    * add，add（int index,item），size(),isEmpty(),remve(index),clear

---

  * 回顾

  * 神兽系统

    * 面向对象（封装，继承，多态），异常，log4j2，List，ArrayList

  * 集合

    * 容器（引用类型）

    * 数组的区别：长度

* 框架图

* Collection（单）

* add，addAll，remove，removeAll，size,isEmpty,toArray,clear,contains,containsAll

* List（有序，可重复）

* get(int index),add

* ArrayList（底层：数组，查询快，增删慢）

* Vector（线程安全）

* LinkedList（底层：双向链表，查询慢，增删快）

* CopyOnWriteArrayList

* Set（无序，不能重复）

* HashSet（底层用HashMap的key）

* Person：equals加上hashcode

* LinkedHashSet(有序)

* TreeSet（排序）

* Person：Comparable--->compareTo

Comparator

* ConcurrentSkipListSet(jdk1.5 并发编程)

* Queue

* ConcurrentLinkedQueue(jdk1.5 并发编程)

* ConcurrentLinkedDeque(jdk1.5 并发编程)

* 存储数据：jdk1. 5 --- LinkedList

* Map(双)

* Iterator,增强型for

* 泛型

* 预定义未知类型

* 把运行时异常转换编译时异常

* 避免强制类型转换

* 代码的复用

* 语法

* 泛型类，泛型方法，泛型接口，通配符，上限 extends，下限 super

public static <T extends Comparable<T>> int statictisGreaterCount(T[] items,T item){

// compareTo--->Entity-(Comparable)

}

* 斗地主

* List<Card> list=new ArraryList<Card>();

* Collections.shuffle();

* 能够理解数据结构的概述

* 数据结构是计算机存储、组织数据的方式。

* 学好数据结构可以编写出更加优雅，更加有效率的的代码

* 能够理解数据结构之栈

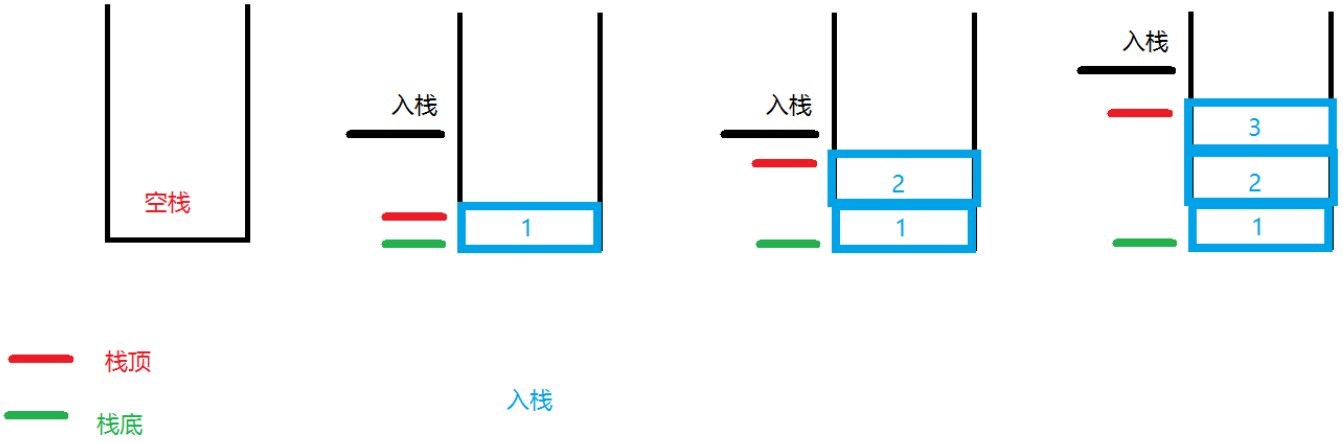
* 栈英文为（stack）

* 栈是一个后进先出的有序列表。（First In Last Out:FILO）

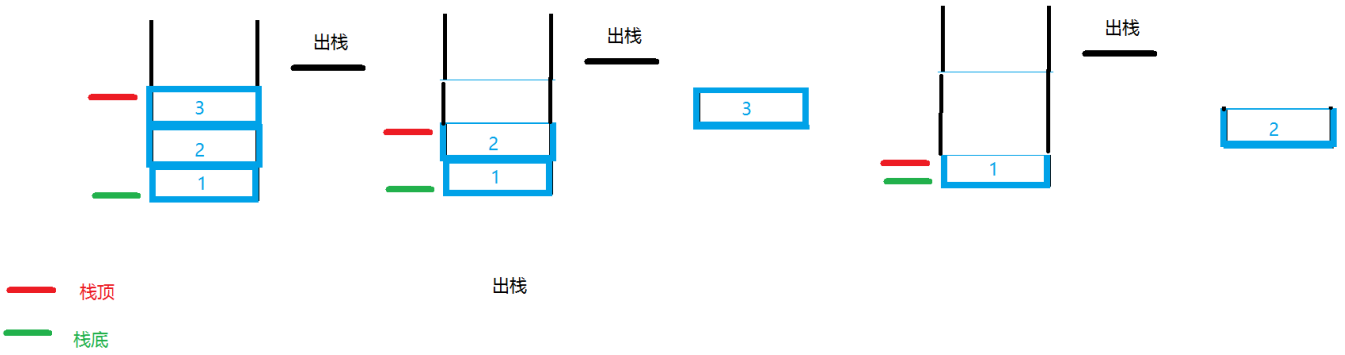
* 弹夹的例子（压栈，弹栈）

* 栈是运算受限的线性表，其限制是仅允许在标的一端进行插入和删除操作，不允许在其他任何位置进行添加、查找、删除等操作。 允许插入和删除的一端，变化的一端称为栈顶（Top），另外固定一端称为栈底（Bottom）。

栈：后进先出



栈：后进先出

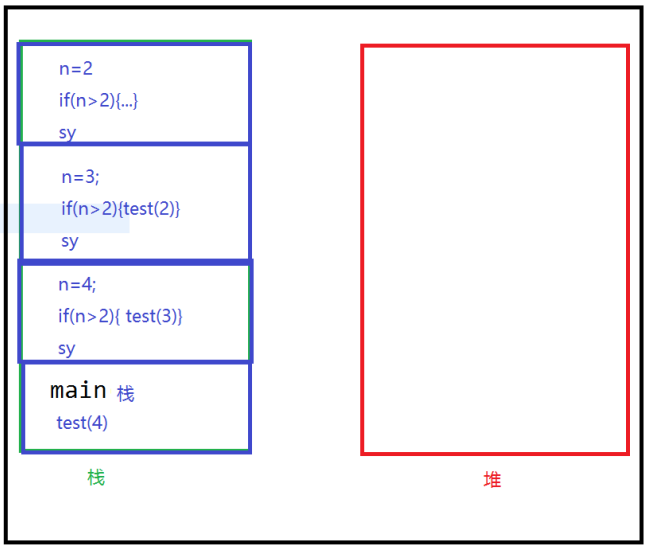



```
Stack
* 代码：
   public static void main(String[] args) {
        Stack<String> stack=new Stack<String>();
        stack.push("刘备");
        stack.push("关羽");
        stack.push("张飞");
        System.out.println(stack);
        String value = stack.pop();//假如栈里面没有数据进行弹栈：EmptyStackExcepti
        System.out.println(value);
        System.out.println(stack);
   }
* 结果：
 [刘备, 关羽, 张飞]
```

```
 张飞
 [刘备, 关羽]

 * 方法递归
   test1(){
     //...:假如没有终止条件：就会报异常：StackOverflowError
     test1();
     //...
   }

* 方法递归调用
  * 一种情况
   public static void main(String[] args) {
       test(4);
   }

   // 方法递归调用
   private static void test(int n) {
       if(n>2) {
           test(n-1);
       }
       System.out.println("n="+n);
   }
结果：
   n=2
   n=3
   n=4
 * 第二种情况
 public static void main(String[] args) {
       test(4);
   }

   // 方法递归调用
   private static void test(int n) {
       if(n>2) {
           test(n-1);
       }else {
           System.out.println("n="+n);
       }
   }
```

```
结果：
   n=2
```

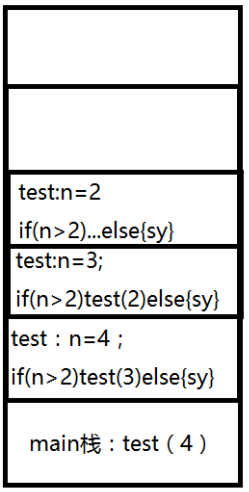
```java
public static void main(String[] args) {
    test(4);
}

// 方法递归调用
private static void test(int n) {
    if(n>2) {
        test(n-1);
    }
    System.out.println("n="+n);
}
```



方法压栈的过程

剪贴板 图像 工具 形状 颜色

```java
public static void main(String[] args) {
    test(4);
    System.out.println("程序结束");
}

public static void test(int n) {
    if(n>2) {
        test(n-1);
    }else {
        System.out.println(n);
    }
}
```



栈

* 温馨提示：

  * 递归需要遵守规则

    * 执行一个方法时，就会创建独立的空间（栈空间）

      * 局部变量是独立的

* 必须有条件，否则就是无限递归(死龟)，会出现StackOverflowError

* 当一个方法执行完毕或者遇到return，遵守谁调用，就将结果返回给谁。

* 能够理解数据结构之队列

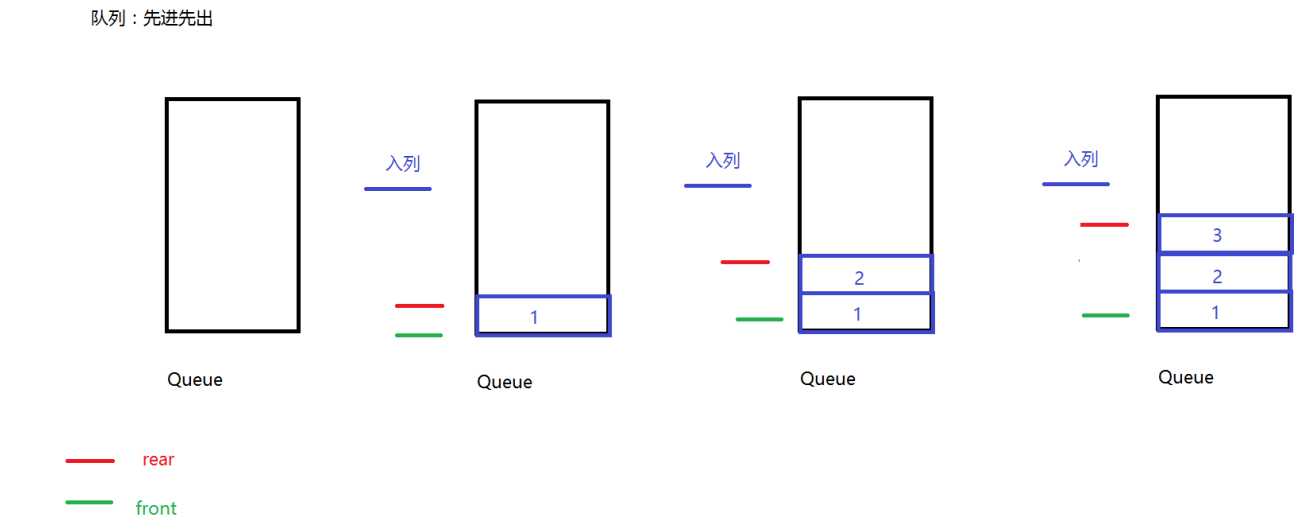
* 队列的英文为（queue）

* 队列是先进先出有序列表。（Fisrt In First Out）：FIFO

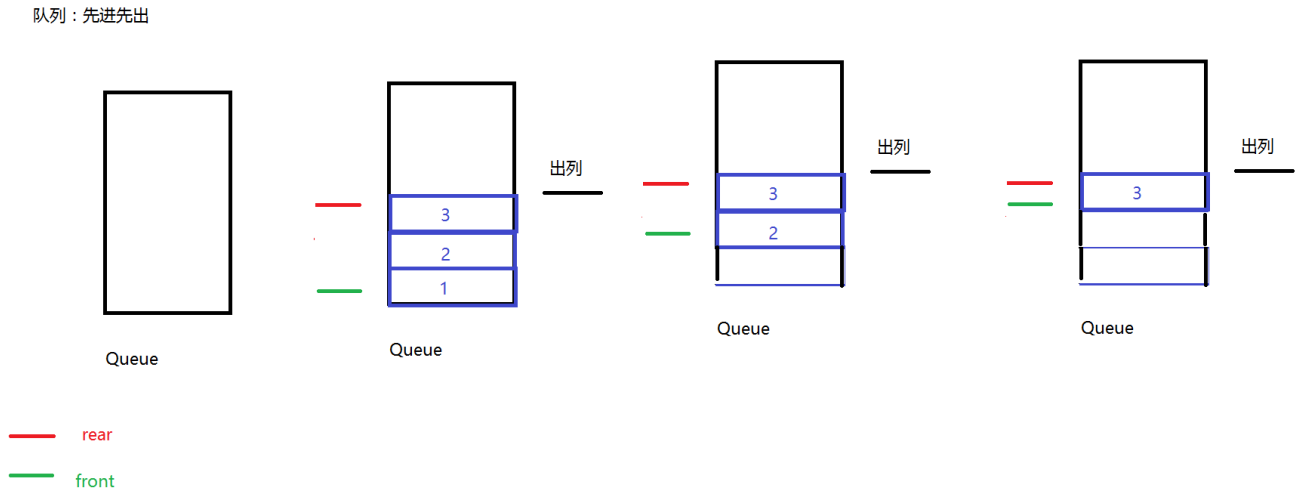
* 车过高速公路消费站

* 队列是一种运算受限的线性表，其限制是仅允许在表的一端进行插入， 而在表的另一端进行删除。

* 入列



* 出列



案例

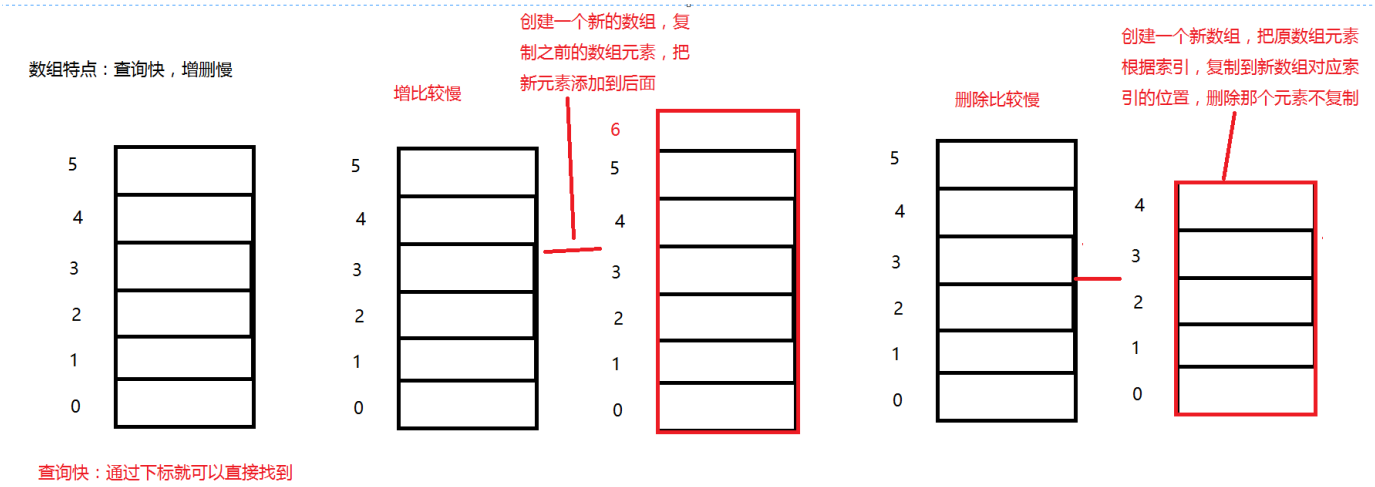
```
* LinkedList 实现队列的接口
public static void main(String[] args) {
        Queue<String> queue=new LinkedList<String>();
        queue.add("刘备");
        queue.add("关羽");
        queue.add("张飞");
        System.out.println(queue);
        String value=queue.poll();
        System.out.println(value);
        System.out.println(queue);
    }
 结果:
[刘备, 关羽, 张飞]
刘备
[关羽, 张飞]
```

* 能够理解数据结构之数组

  * 数组英文名字：Array

  * 数组是有序的元素序列，数组是在内存中开辟一段连续的空间，并在此空间存放元素

  * 数组特点：长度固定，查询快，增删慢

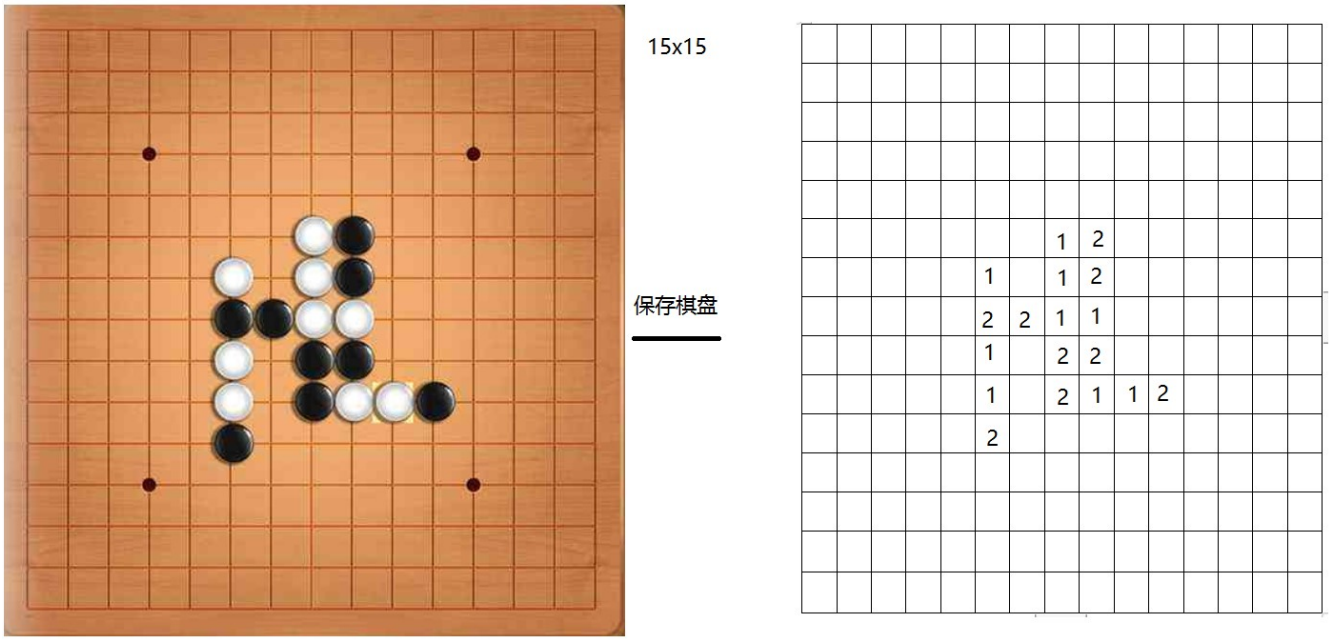


  * 稀疏数组

* 稀疏数组

  * 概述：当一个数组中大部分元素为0或者同一个值时，可以使用稀疏数组保存数组。

* 五子棋器存盘退出和续上盘的功能

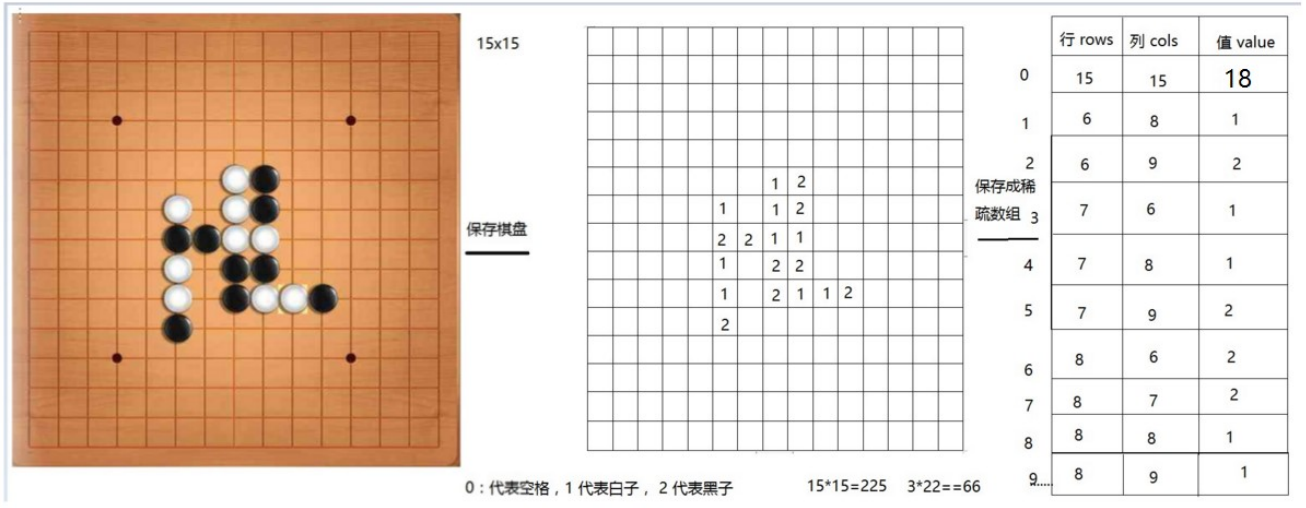


* 分析：该二维数组记录了很多0的数据，因此记录很多无意义，此时可以用稀疏数组保存

* 如何用稀疏数组保存？

  * 记录数组一共有几行几列，有多少个不同的值。

  * 把具有不同值的元素的行列及值记录到一个小规模数组中，从而缩小规模。



* 二维数组转换成稀疏数组

```
1 * 思路
```

```
 * 遍历二维数组得到非0有效数据个数
 * 创建二维稀疏数组
 * 给二维稀疏第一行数组赋值
 * 遍历二维数组，把非0有效数据存储到二维稀疏数组

 /**
     * @param arr:原始二维数组
     * @return 稀疏数组
     */
    public static int[][] formatArrayToSparse(int[][] arr){
        if(arr==null) {
            throw new IllegalArgumentException("arr is null");
        }
//         * 遍历二维数组得到非0有效数据个数
        int sum=0;
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr[i].length; j++) {
                if(arr[i][j]!=0) {
                    sum++;
                }
            }
        }
//         * 创建二维稀疏数组
        int[][] sparseArr=new int[sum+1][3];
//         * 给二维稀疏数组赋值
        sparseArr[0][0]=arr.length;// 行
        sparseArr[0][1]=arr[0].length;// 列
        sparseArr[0][2]=sum;
//         * 遍历二维数组，把非0有效数据存储到二维稀疏数组
        int count=0;
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr[i].length; j++) {
                if(arr[i][j]!=0) {
                    count++;
                    sparseArr[count][0]=i;// 行
                    sparseArr[count][1]=j;// 列
                    sparseArr[count][2]=arr[i][j];// 赋值
                }
            }
        }
```

```
        return sparseArr;
    }
```

* 稀疏数组转换成二维数组

```
 * 思路
  * 读取稀疏数组第一行，创建二维数组
  * 再读接下来稀疏数组，赋值给二维数组
  /**
     * @param sparseArr：稀疏数组
     * @return 转换为二维数组
     */
    public static int[][] parseSparseArray(int[][] sparseArr){
        if(sparseArr==null) {
            throw new IllegalArgumentException("sparseArr is null");
        }
//        * 读取稀疏数组第一行，创建二维数组
        int rows=sparseArr[0][0];
        int cols=sparseArr[0][1];
        int[][] arr=new int[rows][cols];
//        * 再读接下来稀疏数组，赋值给二维数组
        for (int i = 1; i < sparseArr.length; i++) {
            arr[sparseArr[i][0]][sparseArr[i][1]]=sparseArr[i][2];
        }
        return arr;
    }
```

* 测试

```
public static void main(String[] args) {
        int[][] arr=new int[15][15];
        arr[6][8]=1;
        arr[6][9]=2;
        arr[7][6]=1;
        arr[7][8]=1;
```

```
        arr[7][9]=2;
        arr[8][6]=2;
        arr[8][7]=2;
        arr[8][8]=1;
        arr[8][9]=1;
        //棋盘数太多了，省略一部分
        System.out.println("原数组：");
        printArr(arr);
        int[][] sparseArr = SparseArrayUtils.formatArrayToSparse(arr);
        System.out.println("稀疏数组：");
        printArr(sparseArr);
        System.out.println("稀疏数组转换为原数组：");
        int[][] arrNew = SparseArrayUtils.parseSparseArray(sparseArr);
        printArr(arrNew);
    }

    /**
     * @param arr
     * 打印二维数组
     */
    private static void printArr(int[][] arr) {
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr[i].length; j++) {
                System.out.printf("%d ",arr[i][j]);
            }
            System.out.println();
        }
    }

结果：
原数组：
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 2 0 0 0 0 0
0 0 0 0 0 0 1 0 1 2 0 0 0 0 0
0 0 0 0 0 0 2 2 1 1 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
稀疏数组：
15 15 9
6  8  1
6  9  2
7  6  1
7  8  1
7  9  2
8  6  2
8  7  2
8  8  1
8  9  1
稀疏数组转换为原数组：
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 2 0 0 0 0 0
0 0 0 0 0 0 0 1 0 1 2 0 0 0 0 0
0 0 0 0 0 0 0 2 2 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

* 能够理解数据结构之链表（预习）

* 链表英文为：Linked List

* 链表是由一系列结点node（链表中每一个元素称为结点）组成，结点可以在运行时动态生成。

每个结点包括两个部分：一个是存储数据元素的数据域，另一个是存储下一个结点地址的指针域。

我们常说链表有两种：单向链表和双向链表。

* 手拉手

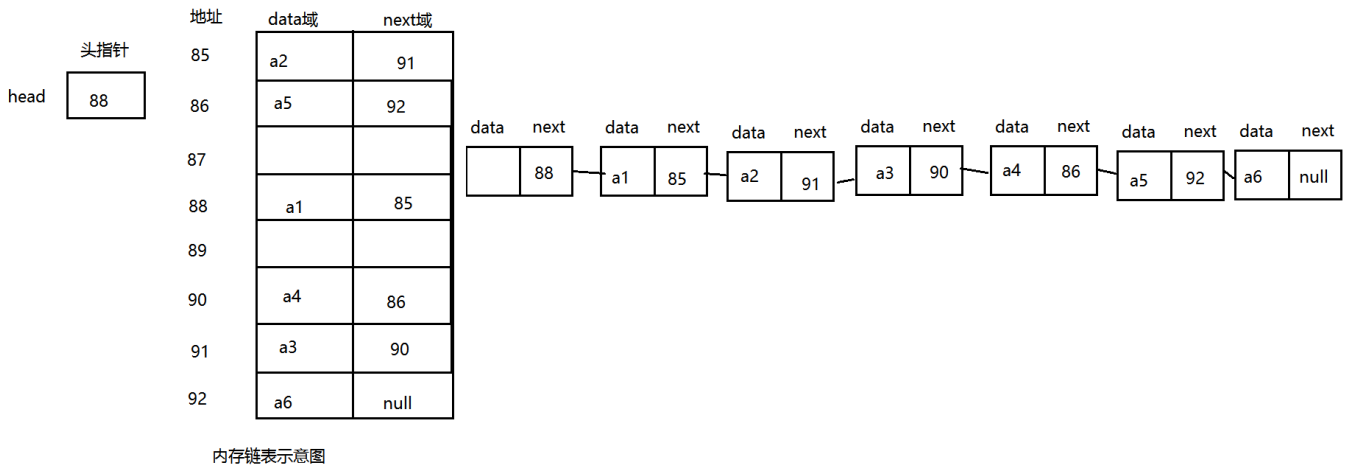
* 链表可以带头节点也不带，看具体的需求

* 链表各个节点不一定是连续存储的。

* 多个结点之间，通过地址进行连接。

* 特点：查询慢，增删快。

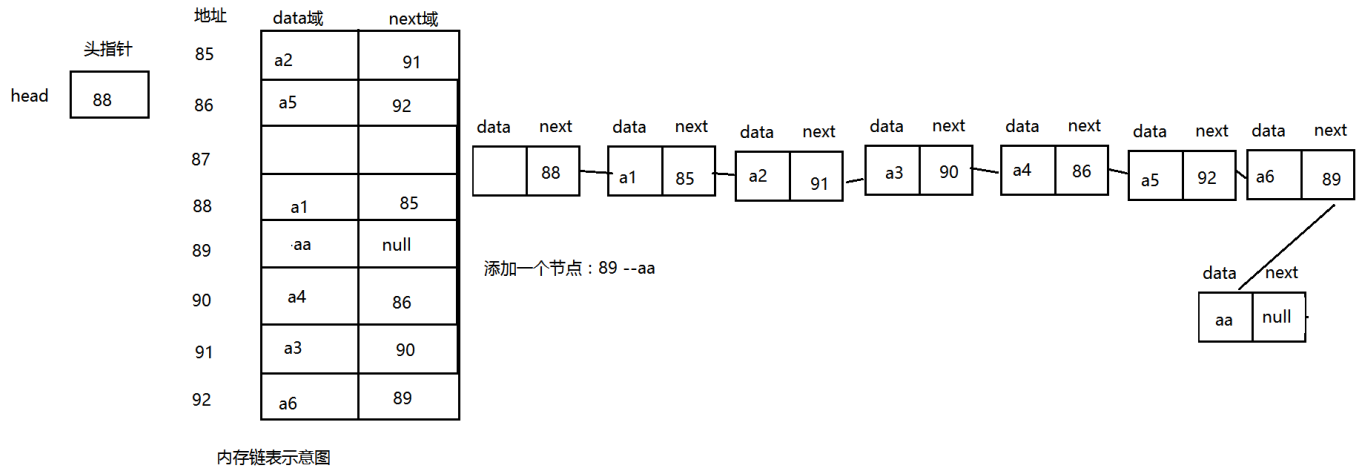
* 单链表示意图



内存链表示意图

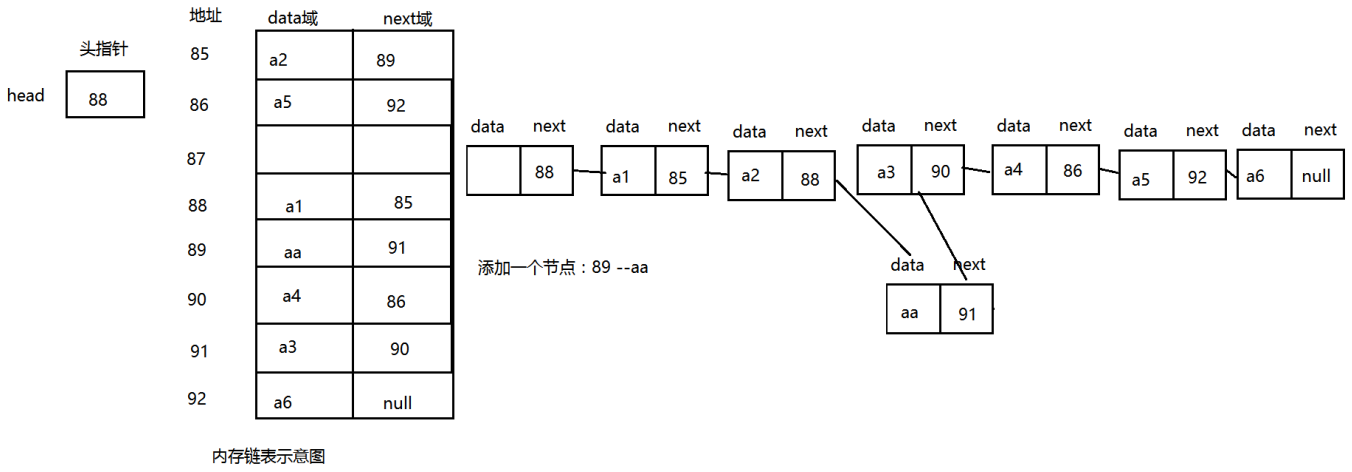
* 增删快：画图解释

* 添加在尾部



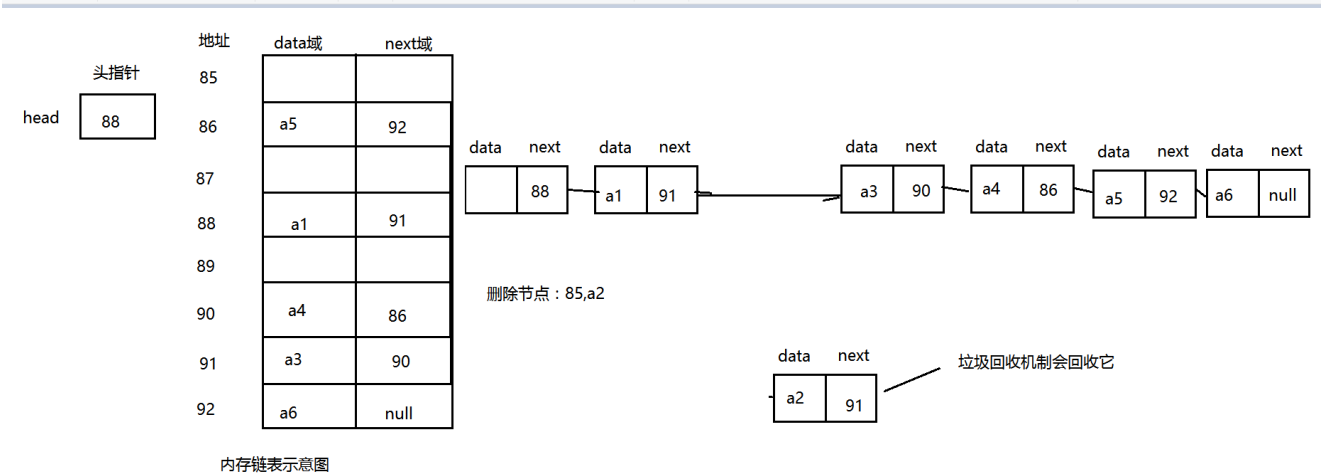
内存链表示意图

* 添加在中间



内存链表示意图

· * 删除

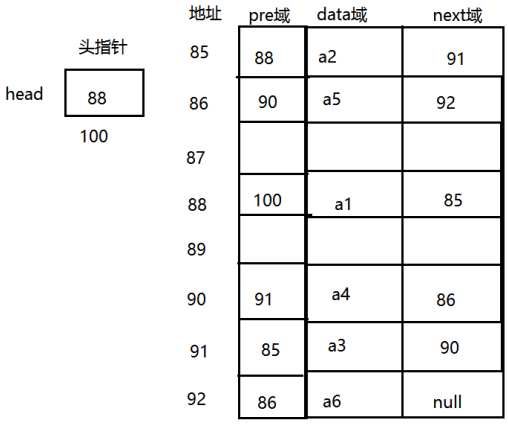


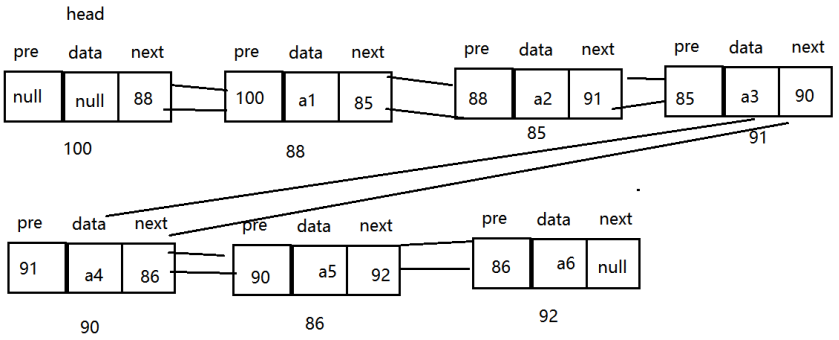
内存链表示意图

* 双向链表

* 单向链表只能往一个方向查找。

* 双向链表可以双向查找





* 能够实现自定义LinkedList