队列

2020年5月12日

12:30

1) 队列是一个有序列表，可以用数组或是链表来实现。

2) 遵循先入先出的原则。

即：先存入队列的数据，要先取出，后存入的要后取出

计算机生成了可选文字:
MaxSize-l
3
2
1
0
Queue
MaxSize-l
MaxSize-l
<．r产3
<．丘on忡2
<-rear=-1
<·丘ont宀1
3
2
1
0
Queue
<-rear=3
<-front—-l
3
2
1
0
Queue

* 数组模拟队列思路

计算机生成了可选文字:
队列本身是有序列表，若使用数组的结构来存储队列的数据，则队列数组的声明如下图，其中maxSize是该队
列的最大容量。
因为队列的输出、输入是分别从前后端来处理，因此需要两个变量front及rear分别记录队列前后端的下标，
front会随着数据输出而改变，而rear则是随着数据输入而改变，如图所示：
MaxSize-l
3
2
0
MaxSize-1
MaxSize-1
3
2
0
<．rea产3
<-front=-l
3
2
0
<·r产3
<珩佣忡2
<-rear=．1
Queue<-front—-l
Queue
当我们将数据存入队列时称为”addQueue”，addQueue的处理需要有两个步骤：思路分析
1)将尾指针往后移：rear+l，当front一rear〖空〗
2）若尾指针rear小于队列的最大下标maxSize-l,则将数据存入rear所指的数组元素中，否则无法存入数据。
==maxSize-1[队列满]
rear

|  |  |
| --- | --- |
|  | package 数据结构;    import java.util.Scanner;    //数组实现队列  public class ArrayQueue1 {  public static void main(String[] args) {  ArrayQueue1 queue1 = new ArrayQueue1(3);  char key; //接收用户输入  Scanner scanner = new Scanner(System.in);  boolean loop = true;  while (loop) {  System.out.println("s(show): 显示队列");  System.out.println("e(exit): 退出程序");  System.out.println("a(add): 添加数据到队列");  System.out.println("g(get): 从队列取出数据");  System.out.println("h(head): 查看队列头的数据");  key = scanner.next().charAt(0);//接收一个字符  switch (key) {  case 's':  queue1.showQueue();  break;  case 'a':  System.out.println("输出一个数");  int value = scanner.nextInt();  queue1.addQueue(value);  break;  case 'g': //取出数据  try {  int res = queue1.getQueue();  System.out.printf("取出的数据是%d\n", res);  } catch (Exception e) {  System.out.println(e.getMessage());  }  break;  case 'h': //查看队列头的数据  try {  int res = queue1.headQueue();  System.out.printf("队列头的数据是%d\n", res);  } catch (Exception e) {  System.out.println(e.getMessage());  }  break;  case 'e': //退出  scanner.close();  loop = false;  break;  }  System.out.println("程序退出！");  }  }    private int maxSize;//队列最大存储元素  private int front;//指向队列头部的前一个位置  private int rear;//指向队列的尾部  private int arr[];//存储队列元素的数组    //构造方法  public ArrayQueue1(int maxSize) {  this.maxSize = maxSize;  this.front = -1;  this.rear = -1;  this.arr = new int[maxSize];  }    //判断队列是否为满  public boolean judgeFull() {  return rear == maxSize - 1;  }    //判断队列是否为空  public boolean judgeEmpty() {  return front == rear;  }    //添加数据到队列  public void addQueue(int n) {  if (judgeFull()) {  System.out.println("队列满，不能添加元素");  return;  } else {  rear++;  arr[rear] = n;  }  }    //取出队列元素  public int getQueue() {  if (judgeEmpty()) {  throw new RuntimeException("队列空，不能取数据");  } else {  front++;  return arr[front];  }  }    //显示队列所有元素  public void showQueue() { // 遍历  if (judgeEmpty()) {  System.out.println("队列空的，没有数据");  return;  }  for (int i = 0; i < arr.length; i++) {  System.out.printf("arr[%d]=%d\n", i, arr[i]);  }  }    // 显示队列的头数据， 注意不是取出数据  public int headQueue() {  if (judgeEmpty()) {  throw new RuntimeException("队列空的，没有数据~~");  }  return arr[front + 1];  }  } |

* 数组模拟环形队列

计算机生成了可选文字:
分析说明：
1）尾索引的下一个为头索引时表示队列满，即将队列容量空出一个作为约定，这个在做判断队列满的
时候需要注意(rear+1)％maxSize==front满]
2）rear==front[空
3）分析示意图：
3
2
0
MaxSize-l
3
2
Queue<-front—-l
Ma、S01
2
<•front=2
思路如下：
1．fr。nt变量的含义做一个调整：fr。nt就指向队列的第一个元素，也就是说arr[frontl就罡队列的第一个元素
front的初始值=0
2．rear变量的含义做一个调整：艹r指向队列的最后一个元素的后一个位乱因为磊望仝出一个仝间做为约定，
rear的初始值=0
3．当队列满时，条件是(rear+1}%maxSize=front〖腐〗
4．对队列为窒的条件，rear==front仝
5．当我们这样分析，队列中有效的据的个(rear+maxSize一front)％maxSize//rear=Ifront=o
6．我们就可以在原来的队列上修改得到，一个环形队列

|  |  |
| --- | --- |
|  | package 数据结构;  import java.util.Scanner;  //数组实现环形队列  public class ArrayQueue2 {  public static void main(String[] args) {  ArrayQueue2 queue1 = new ArrayQueue2(3);  char key; //接收用户输入  Scanner scanner = new Scanner(System.in);  boolean loop = true;  while (loop) {  System.out.println("s(show): 显示队列");  System.out.println("e(exit): 退出程序");  System.out.println("a(add): 添加数据到队列");  System.out.println("g(get): 从队列取出数据");  System.out.println("h(head): 查看队列头的数据");  key = scanner.next().charAt(0);//接收一个字符  switch (key) {  case 's':  queue1.showQueue();  break;  case 'a':  System.out.println("输出一个数");  int value = scanner.nextInt();  queue1.addQueue(value);  break;  case 'g': //取出数据  try {  int res = queue1.getQueue();  System.out.printf("取出的数据是%d\n", res);  } catch (Exception e) {  System.out.println(e.getMessage());  }  break;  case 'h': //查看队列头的数据  try {  int res = queue1.headQueue();  System.out.printf("队列头的数据是%d\n", res);  } catch (Exception e) {  System.out.println(e.getMessage());  }  break;  case 'e': //退出  scanner.close();  loop = false;  break;  }  System.out.println("程序退出！");  }  }  private int maxSize;//队列最大存储元素  private int front;//指向队列头部的前一个位置  private int rear;//指向队列的尾部  private int arr[];//存储队列元素的数组    //构造方法  public ArrayQueue2(int maxSize) {  this.maxSize = maxSize;  this.front = 0;  this.rear = 0;  this.arr = new int[maxSize];  }    //判断队列是否为满  public boolean judgeFull() {  return (rear + 1) % maxSize == front;  }    //判断队列是否为空  public boolean judgeEmpty() {  return front == rear;  }    //添加数据到队列  public void addQueue(int n) {  if (judgeFull()) {  System.out.println("队列满，不能添加元素");  return;  } else {  arr[rear] = n;  rear=(rear+1)%maxSize;  }  }    //取出队列元素  public int getQueue() {  if (judgeEmpty()) {  throw new RuntimeException("队列空，不能取数据");  } else {  int value=arr[front];  front=(front+1)%maxSize;  return value;  }  }    //显示队列所有元素  public void showQueue() { // 遍历  if (judgeEmpty()) {  System.out.println("队列空的，没有数据");  return;  }  for (int i = front; i < front+(maxSize+rear-front)%maxSize; i++) {  System.out.printf("arr[%d]=%d\n", i%maxSize, arr[i%maxSize]);  }  }    // 显示队列的头数据， 注意不是取出数据  public int headQueue() {  if (judgeEmpty()) {  throw new RuntimeException("队列空的，没有数据~~");  }  return arr[front];  }  } |