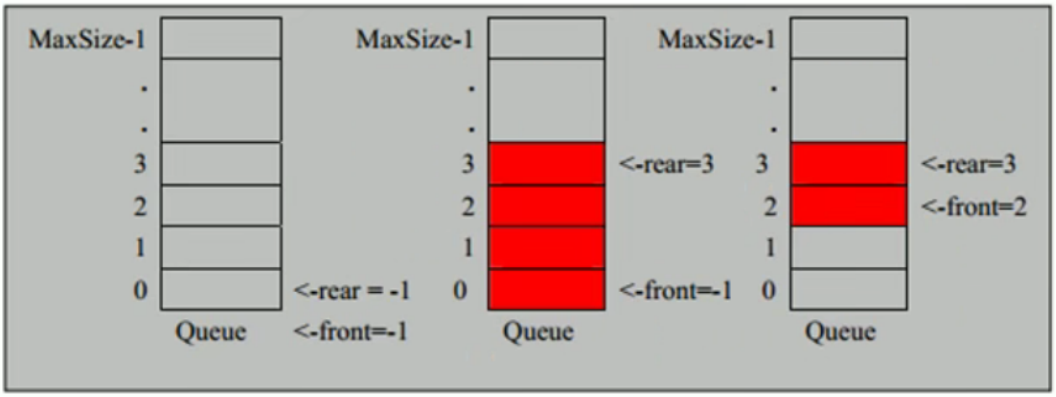
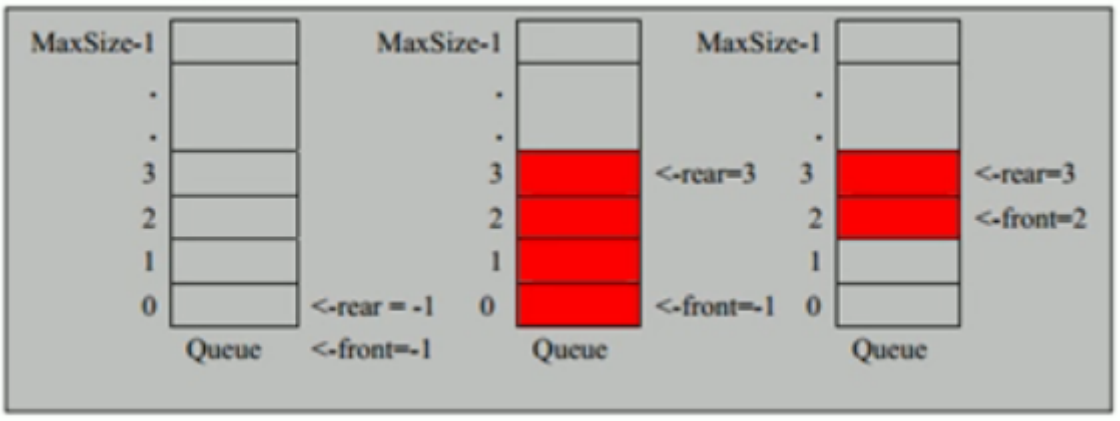
队列介绍

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
3. 示意图



数组模拟队列

1. 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
2. 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：



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 + 1，当 front == rear时，队列为空

2. 若尾指针 rear 小于队列的最大下标 maxSize - 1，则将数据存入 rear 所指的数组元素中，否则无法存入数据。rear == maxSize - 1，队列满

代码实现

```
import java.util.Scanner;

/**
 * 使用数组模拟队列
 *
 * @program: data-structure-algorithm
 * @author: ke
 * @create: 2021-09-07 11:20
 **/
public class ArrayQueueDemo {

    public static void main(String[] args) {
        // 创建一个队列
        ArrayQueue queue = new ArrayQueue(3);
        // 接受用户输入
        char key = ' ';
        // 扫描器
        Scanner scanner = new Scanner(System.in);
        boolean loop = true;
        // 输出一个菜单
        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");
            System.out.println("g(get): 取出数据");
            System.out.println("h(head): 头部数据");
            // 接收一个字符串的第一个字符
            key = scanner.next().charAt(0);
            switch (key) {
                case 's':
                    queue.showQueue();
                    break;
                case 'a':
                    System.out.println("请输入一个数字");
                    int value = scanner.nextInt();
                    queue.addQueue(value);
                    break;
                case 'g':
                    try {
                        int res = queue.getQueue();
                        System.out.printf("取出的数据是%d\n", res);
                    } catch (Exception e) {
                        System.out.println(e.getMessage());
                    }
                    break;
                case 'h':
                    try {
                        int res = queue.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':
                    scanner.close();
                    loop = false;
                    break;
                default:
                    break;
            }
        }
        System.out.println("程序退出");
    }
}

class ArrayQueue {
    /**
     * 表示数组的最大容量
     */
    private int maxSize;
    /**
     * 队列头
     */
    private int front;
    /**
     * 队列尾
     */
    private int rear;
    /**
     * 该数据用于存放数据，模拟队列
     */
    private int[] arr;

    /**
     * 创建队列的构造器
     *
     * @param arrMaxSize
     */
    public ArrayQueue(int arrMaxSize) {
        maxSize = arrMaxSize;
        arr = new int[maxSize];
        // 指向队列头部，分析出 front 时指向队列头的前一个位置
        front = -1;
        // 指向队列尾部，队列尾的数据（即队列最后一个数据）
        rear = -1;
    }

    /**
     * 判断队列是否满
     *
     * @return
     */
    public boolean isFull() {
        return rear == maxSize - 1;
    }

    /**
```

```
     * 判断队列是否为空
     *
     * @return
     */
    public boolean isEmpty() {
        return rear == front;
    }

    /**
     * 添加数据到队列
     *
     * @param n
     */
    public void addQueue(int n) {
        // 判断队列是否满
        if (isFull()) {
            System.out.println("队列满，不能加入数据");
            return;
        }
        // 让 rear 后移
        rear++;
        arr[rear] = n;
    }

    /**
     * 获取队列的数据，出队列
     *
     * @return
     */
    public int getQueue() {
        if (isEmpty()) {
            // 抛出异常
            throw new RuntimeException("队列空，不能取数据");
        }
        // front 后移
        front++;
        return arr[front];
    }

    /**
     * 显示队列的所有数据
     */
    public void showQueue() {
        if (isEmpty()) {
            System.out.println("空队列，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
     *
     * @return
```

```
     */
    public int headQueue() {
        if (isEmpty()) {
            throw new RuntimeException("空队列，没有数据");
        }
        return arr[front + 1];
    }
}
```

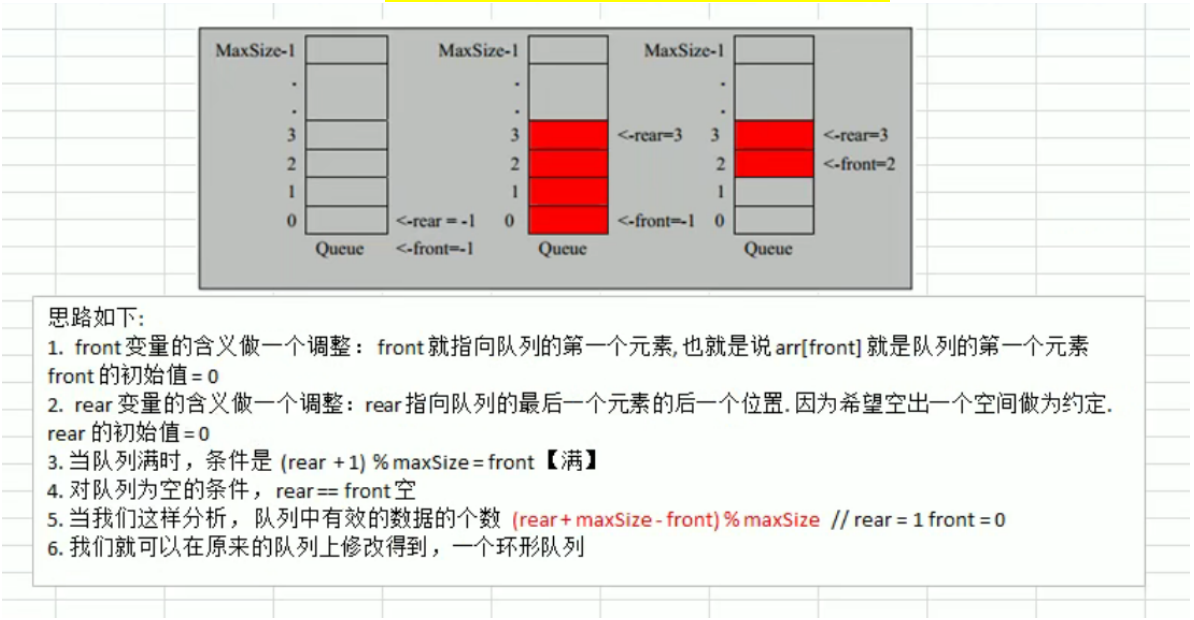
问题分析并优化

1. 目前数组使用一次就不能用，没有达到复用的效果
2. 将这个数组使用算法，改进成一个环形的队列 取模：%

数组模拟环形队列

对前面的数组模拟队列的优化，充分利用数组，因此将数组看做是一个环形的（通过取模的方式来实现即可）

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 表示队列已满
2. rear == front 表示队列为空，多出一个空容量可以区分空队列与满队列



代码实现

```
import java.util.Scanner;

/**
 * 使用数组模拟环形队列
 *
 * @program: data-structure-algorithm
```

```
 * @author: ke
 * @create: 2021-09-07 16:27
 **/
public class CircleArrayQueueDemo {

    public static void main(String[] args) {
        // 创建一个环形队列，这里设置 4 ，其队列的有效数据最大是 3
        CircleArray queue = new CircleArray(4);
        // 接受用户输入
        char key = ' ';
        // 扫描器
        Scanner scanner = new Scanner(System.in);
        boolean loop = true;
        // 输出一个菜单
        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");
            System.out.println("g(get): 取出数据");
            System.out.println("h(head): 头部数据");
            // 接收一个字符串的第一个字符
            key = scanner.next().charAt(0);
            switch (key) {
                case 's':
                    queue.showQueue();
                    break;
                case 'a':
                    System.out.println("请输入一个数字");
                    int value = scanner.nextInt();
                    queue.addQueue(value);
                    break;
                case 'g':
                    try {
                        int res = queue.getQueue();
                        System.out.printf("取出的数据是%d\n", res);
                    } catch (Exception e) {
                        System.out.println(e.getMessage());
                    }
                    break;
                case 'h':
                    try {
                        int res = queue.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':
                    scanner.close();
                    loop = false;
                    break;
                default:
                    break;
            }
        }
        System.out.println("程序退出");
    }
```

```
}

class CircleArray {
    /**
     * 表示数组的最大容量
     */
    private int maxSize;
    /**
     * 队列头，front 就指向队列的第一个元素，也就是 arr[front]，front 的初始值 = 0
     */
    private int front;
    /**
     * 队列尾，rear 指向队列的最后一个元素的后一个位置。因为希望空出一个空间作为约定 maxSize -
1，rear 的初始值 = 0
     */
    private int rear;
    /**
     * 该数据用于存放数据，模拟队列
     */
    private int[] arr;

    public CircleArray(int arrMaxSize) {
        maxSize = arrMaxSize;
        arr = new int[maxSize];
    }

    /**
     * 判断队列是否满
     *
     * @return
     */
    public boolean isFull() {
        return (rear + 1) % maxSize == front;
    }

    /**
     * 判断队列是否为空
     *
     * @return
     */
    public boolean isEmpty() {
        return rear == front;
    }

    /**
     * 添加数据到队列
     *
     * @param n
     */
    public void addQueue(int n) {
        // 判断队列是否满
        if (isFull()) {
            System.out.println("队列满，不能加入数据");
            return;
        }
        arr[rear] = n;
        // 将 rear 后移，这里必须考虑取模
```

```
        rear = (rear + 1) % maxSize;
    }

    /**
     * 获取队列的数据，出队列
     *
     * @return
     */
    public int getQueue() {
        if (isEmpty()) {
            // 抛出异常
            throw new RuntimeException("队列空，不能取数据");
        }
        /*
         * 这里需要分析出 front 是指向队列的第一个元素
         * 1.先把 front 对应的值保留到一个临时变量
         * 2.将 front 后移，考虑取模
         * 3.将临时保存的变量返回
         */
        int value = arr[front];
        front = (front + 1) % maxSize;
        return value;
    }

    /**
     * 显示队列的所有数据
     */
    public void showQueue() {
        if (isEmpty()) {
            System.out.println("空队列，没有数据");
            return;
        }

        /*
         * 思路：从 front 开始遍历，遍历多少个元素
         */
        for (int i = front; i < front + size(); i++) {
            System.out.printf("arr[%d]=%d\n", i % maxSize, arr[i % maxSize]);
        }
    }

    /**
     * 求出当前队列有效数据的个数
     *
     * @return
     */
    public int size() {
        /*
         * rear = 1
         * front = 0
         * maxSIze = 3
         */
        return (rear + maxSize - front) % maxSize;
    }

    /**
     * 显示队列的头数据
```

```
     *
     * @return
     */
    public int headQueue() {
        if (isEmpty()) {
            throw new RuntimeException("空队列，没有数据");
        }
        return arr[front];
    }
}
```