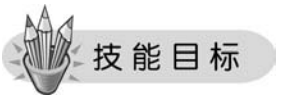
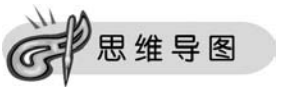
模块 3　队列——医院排队叫号系统



- 理解队列的定义，掌握队列的特征和基本运算。
- 会使用队列的顺序存储结构解决问题。
- 能实现循环队列的各种基本运算。
- 会使用队列的链式存储结构解决问题。
- 能实现链式队列的各种基本运算。



本模块思维导图请扫描右侧二维码。

队列思维导图

像栈一样，队列也是一种线性表。然而，使用队列时，插入在一端进行，而删除则在另一端进行。就像我们平时的排队一样，出队是在队首的位置，而入队则在队尾的位置。

队列最典型的一个功能就是秒杀问题。就像抢火车票或者抢小米手机一样，在整点的时候，大量的请求涌入，如果仅仅依靠服务器来处理，超高的并发量不仅会带给服务器巨大压力，而且有可能出现各种高并发场景下才会出现的问题，比如超卖、事务异常等。

而队列，正是解决这个问题的一把“好手”。通常我们会使用的都是以内存为主的队列系统，它们的特点就是存储非常快。由前端生成的大量请求都存入队列中(入队)，然后在后台脚本中进行处理(出队)。前端只需要返回一个正在处理中，或者正在排队的提示即可，然后后台处理完成后，通知前台显示结果。这样，在一个秒杀场景中基本上就解决高并发的问题了。

本模块将介绍队列的基本概念、运算以及常见的实现方法，并通过一个有趣的案例介绍队列的应用。

3.1　项目描述

医院接诊需要排队叫号，请设计一个软件，模拟排队叫号的场景，如图 3-1 所示。

1. 功能描述

该软件包括三个功能按钮和两个展示就诊人和已诊人的文本框。队列长度默认为 20 人。当用户单击“放号”按钮时，显示可以就诊人的总数；单击“取号”按钮后，将把第 n 个人推入就诊队列中，并放在第 $n-1$ 人的后面，此过程显示在就诊人队列一栏；单击“就诊”按钮，则从就诊人队列的最上方移动 1 人进入已诊人队列，也是放置在该队列的尾部，如图 3-2 所示。

图 3-1　排队挂号场景示意图

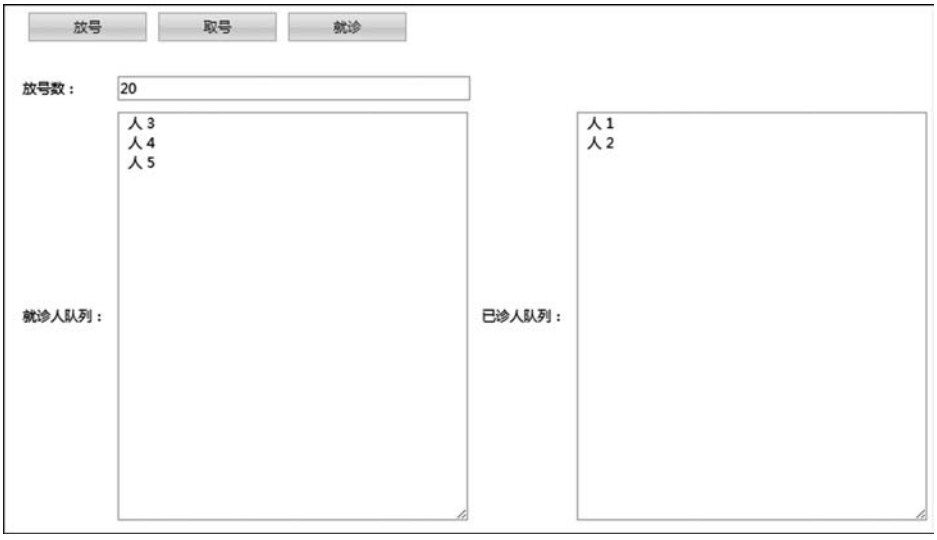


图 3-2　挂号软件界面

具体系统功能需求如下。

(1) 放号：设定当天患者可挂的总号数。

(2) 取号：患者取号。

(3) 就诊叫号：轮到患者就诊时会叫号，排在前面的号先被叫。

(4) 输入错误提示：放号数设定值只能为数字，如输入字符，系统自动将其设置为默认值 20。

2. 设计思路

本项目利用队列结构存储就诊人。“取号”功能对应入队操作，新加的就诊人放在就诊队列尾部，“就诊”功能对应出队操作，即从队首取出当前就诊人，添加到已诊人队列。

3.2 相关知识

观察一下食堂中午排队打饭的场面,可以发现以下一些特点:打饭者整齐地排成一队,组成一个线性表;只有位于队首的同学才能开始打饭,且打完饭即出队;队外的任何人欲打饭,必须从队尾加入队中。

不只在日常生活中,在计算机领域,我们也常常听到“消息队列”“打印队列”等术语。实践证明,以队列的方式组织和操作数据,在许多问题的求解过程中非常有效。

3.2.1 队列的定义

严格地说,与栈一样,队列也是一种特殊的线性表,它仅允许在表的一端(即队首)进行出队(即删除)运算,在表的另一端(即队尾)进行入队(即插入)操作,如图 3-3 所示。因为出队时先入队的元素先出,所以队列又被称为是一种“先进先出”表,简称为 FIFO(fast in first out)。

图 3-3 队列示意图

3.2.2 队列的基本运算

根据实际应用,通常认为,队列应该包含以下一些基本运算。

(1) 队列初始化:置队列为空队。

(2) 判断队列是否为空:若队列为空,则返回 true,否则返回 false。

(3) 求队列的长度:返回队列的元素个数。

(4) 读队首:返回队首元素之值。

(5) 入队:将一个元素插入队尾。

(6) 出队:将队首元素从队列中删除。

在 Java 中,我们用队列的接口 IQueue 表示队列这些功能操作的集合。在后面的例子中,我们将实现这个接口,通过展示不同的实现代码,详细解释顺序队列和链式队列的不同之处。但不管怎样,只要这些类实现了队列的接口,就可以将其称为队列。

队列的接口如下。

【代码 3-1】

```
public interface IQueue {
    public void append(Object obj)throws Exception;                    //入队
    public Object delete()throws Exception;                            //出队
    public Object getFront()throws Exception;                          //取队首
```

```
    public boolean isEmpty();                    //判断队列是否为空
    public int getSize();                        //求队列长度
}
```

3.2.3　顺序队列

队列的顺序存储结构简称为顺序队列。

顺序队列是由一个一维数组和用于指示队首位置与队尾位置的两个变量组成，即

顺序队列＝一维数组＋队首指示＋队尾指示

顺序队列结构如图 3-4 所示。

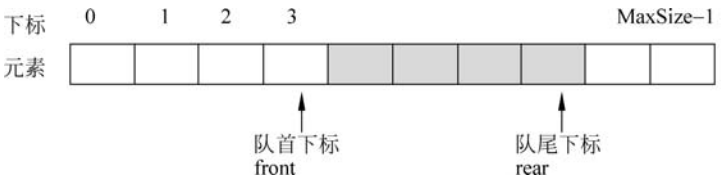


图 3-4　顺序队列的存储结构

这里需要注意，为了实现代码方便，我们通常约定：rear 指向队尾元素在一维数组中的当前位置；front 指向队首元素在一维数组中当前位置的前一个位置。

具体地说，顺序队列用一个 SeqQueue 类实现，其数据类型描述如下。

【代码 3-2】

```
public class CirQueue implements IQueue {
    final int defaultSize = 10;
    int maxSize;
    int front;                   //队首
    int rear;                    //队尾
    Object[] data;               //一维数组
}
```

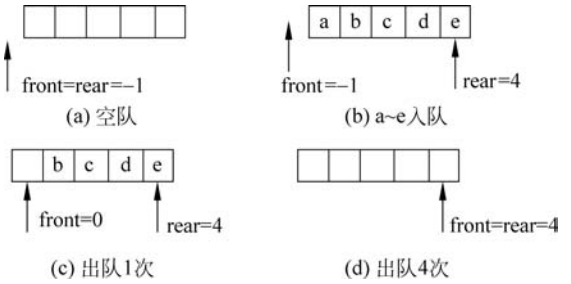
顺序队列用一维数组 data 存放数据，序号为 i 的元素对应数组的下标是 $i-1$，即用 data[$i-1$]表示。队首元素用 data[front+1]表示，队尾元素用 data[rear]表示。

图 3-5 所示是一个 MaxSize 为 5 的队列的动态变化图。图 3-5(a)表示初始的空队列；图 3-5(b)表示入队 5 个元素后队列的状态；图 3-5(c)表示队首元素出队 1 次后队列的状态；图 3-5(d)是队首元素出队 4 次后队列状态。

从图 3-5 中不难看出，队列为空的条件为 front＝＝rear 成立，那么队满条件是不是 rear＝＝MaxSize－1 呢？显然不是。图 3-5(d)也满足这个条件，但却是个空队列。因为无论添加还是删除元素，队首变量和队尾变量始终是向着队列的尾端移动的，这就会使顺序队列产生溢出问题。

(1) 当队列已满，再进行入队操作时，就会产生“上溢出”。

(2) 当队列为空，再进行出队操作时，就会产生“下溢出”。



微课 3-1　顺序队列和循环队列

图 3-5　顺序队列的操作

此外,对图 3-5(c)或(d)进行入队操作时,明明队列还能存放元素,但由于 rear 值已经指示到最大值,因此出现插入异常,这种溢出称为“假溢出”。

为了解决这个问题,充分地利用数组空间,我们将数组的首尾相接,形成一个环状结构,称这种改进的顺序队列为循环队列(circular queue),如图 3-6 所示。

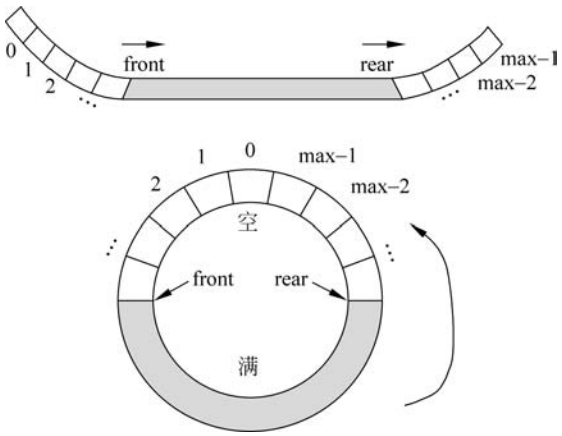


图 3-6　循环队列的逻辑结构

图 3-6 中,将顺序队列的首尾相连,形成一个环。当队尾指示 rear 值为 MaxSize－1 时,若仍然对该队列进行入队操作,则 rear 直接跳到 0。这种变化规律可以用求模运算来实现。

入队操作时,rear 指向下一个位置:

$$rear = (rear + 1) \% \ MaxSize$$

出队操作时,front 指向下一个位置:

$$front = (front + 1) \% \ MaxSize$$

其实,上述算式也可以用下面的伪代码来解释。

【代码 3-3】

```
if (f + 1)< MaxSize              //f 表示 front
    f = f + 1;
else
    f = 0;
```

从图 3-6 可知，初始化时，front 和 rear 的值均为 0。那么队列为空和为满的条件各是什么呢？不难发现，队空的条件是 front＝＝rear；而队满的判断就比较复杂：若入队的速度快于出队的速度，则 rear 的值增加得比 front 快，这样 rear 就有可能赶上 front 的值，此时 front 和 rear 也相等，这样就无法区分队空还是队满。为了解决这个问题，我们常采用这样的办法，空出一个存储空间，让 front 指向队首元素的前一个位置（即 front 指向的位置不存放元素）。

如此约定后，就有如下规则。

初始化时：front＝rear＝0。

循环队列为空的条件：front＝＝rear。

循环队列为满的条件：front＝＝(rear＋1)％MaxSize。

对于该队满条件，也可以用如下伪代码解释。

【代码 3-4】

```
if(rear + 1)< MaxSize
    判断 front 是否等于 rear + 1,是则队满
else
    判断 front 是否等于 0,是则队满
```

对于循环队列的入队和出队操作，可以使用图 3-7 来表示。

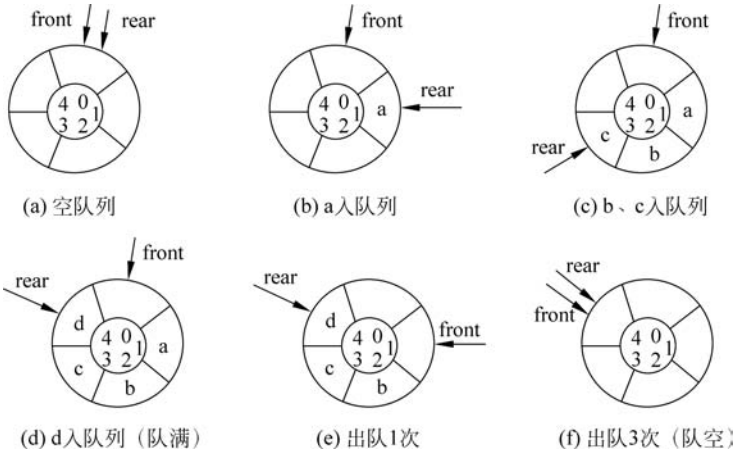


图 3-7　循环队列的操作

以下所讲的顺序队列，我们均将采用循环队列的模式进行存储。从本质上说，循环队列也是顺序队列的一个实现途径。

3.2.4　循环队列

将 3.2.3 小节的 SeqQueue 类名稍作修改，循环队列 CirQueue 的定义如下。

【代码 3-5】

```
public class CirQueue implements IQueue {
    final int defaultSize = 10;
```

```
    int maxSize;
    int front;                  //队首
    int rear;                   //队尾
    Object[ ] data;             //一维数组
}
```

1. 循环队列的基本运算

根据循环顺序队列的运算定义,可实现以下操作。

1) 队列初始化

队列的初始化实现比较简单,通过添加一个 initiate 方法,在该方法中将队首指示 front 和队尾指示 rear 的值设置为 0 即可,同时创建一个用于存储队列中元素的一维数组 data,参数 sz 表示队列的大小。然后在构造函数中调用此方法,代码如下。

【代码 3-6】

```
public class CirQueue implements IQueue{
    final int defaultSize = 20;
    int maxSize;
    int front;                  //队首
    int rear;                   //队尾
    Object[ ] data;             //一维数组

    public CirQueue(){
        initiate(defaultSize);
    }
    public CirQueue(int sz){
        initiate(sz);
    }
    private void initiate(int sz){
        maxSize = sz;
        front = rear = 0;
        data = new Object[sz];
    }
}
```

2) 判断队列是否为空

此处实现接口中的 isEmpty 方法。在判断队列是否为空时,只需比较队首指示 front 和队尾指示 rear 是否相等即可。若相等,则表示队列中不包含任何元素。代码如下。

【代码 3-7】

```
public boolean isEmpty(){
    return front == rear;
}
```

3) 求队列的长度

此处实现接口中的 getSize 方法。队列的长度即为队列中数组元素的个数。长度的计

算按两种情形：rear 值大于 front 值和 rear 值小于 front 值，如图 3-8 所示，左边的图即为第一种情形，右边的图即为第二种情形。对于第一种情形，队列的长度 length＝rear－front；而对于第二种情形，队列的长度 length＝rear＋MaxSize—front。

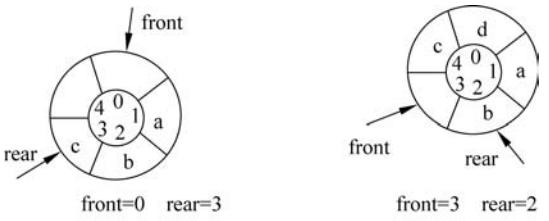


图 3-8　队列长度判断

代码如下。

【代码 3-8】

```
public int getSize() {
    return (rear + maxSize - front) % maxSize;
}
```

4）读队首元素

此处实现接口中的 getFront()方法。根据队首指示 front，可以获取对应的元素，这里分成三类情况，如图 3-9 所示。图 3-9(a)表示进行队空判断，若队空则返回空；图 3-9(b)表示，若 front＋1 小于 MaxSize，则直接返回 front＋1 对应的元素，否则返回 0 对应的元素(求模运算)；图 3-9(c)表示若 front＋1 等于 MaxSize，则返回。

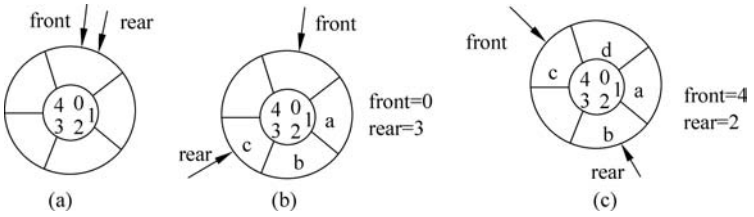


图 3-9　读队首的三种情况

代码如下。

【代码 3-9】

```
public Object getFront() throws Exception{
    if(front == rear)
        return null;
    else
        return data[(front + 1) % maxSize];
}
```

5）入队操作

此处实现接口中的 append()方法。

微课 3-2 队列的入队和出队操作

入队过程的步骤如下。

(1) 队尾指示 rear 值增 1。

(2) 将元素放入队列中由 rear 所指向的位置上。

应该注意的是,进行入队运算时,必须先进行队满检查,以避免错误,同时也应该考虑到当 rear 值达到 MaxSize－1 时,继续增加将使 rear 变为 0,故用到前面所讲的求模运算,代码如下。

【代码 3-10】

```
public void append(Object obj) throws Exception {
    if(front == (rear + 1) % maxSize)
        throw new Exception("队列已满!");
    else {
        rear = (rear + 1) % maxSize;
        data[rear] = obj;
    }
}
```

6) 出队操作

此处实现接口中的 delete()方法。将元素出队就是删除队首指示所对应的元素,其步骤如下。

(1) 获取队首指示的元素。

(2) 将队首指示 front 增值 1。

此外也应注意对队列是否为空进行判断,代码如下。

【代码 3-11】

```
public Object delete() throws Exception{
    if(rear == front)
        return null;
    else{
        Object e = data[(front + 1) % maxSize];
        front = (front + 1) % maxSize;
        return e;
    }
}
```

7) 打印队列中所有元素

此方法非接口中定义的功能,但可以在 CirQueue 类中进行扩展,代码如下。

【代码 3-12】

```
public void print(){
    int i = front;
    while(i!= rear){
        System.out.print(data[(i + 1) % maxSize] + " ");
        i = (i + 1) % maxSize;
    }
    System.out.println();
}
```

要测试上述这些方法，可以编写如下测试端代码，相关结果如图 3-10 所示。

【代码 3-13】

```
public static void main(String[ ] args)throws Exception{
    //创建一个循环队列
    CirQueue cq = new CirQueue();
    //判断队列是否为空
    System.out.println("队列是否为空" + cq.isEmpty());
    //入队操作,1～10 依次入队
    for(int i = 0;i < 10;i++){
        cq.append(i + 1);
    }
    //打印队中元素
    cq.print();
    //显示队首元素
    System.out.println("队首元素为" + cq.getFront());
    //出队 5 次,并显示每一次的出队元素
    for(int i = 0;i < 5;i++){
        System.out.println("出队元素:" + cq.delete());
    }
    //显示队列长度
    System.out.println("队列长度" + cq.size());
}
```

图 3-10　测试端代码运行结果

2. 循环队列的动手实践

1）实训目的

掌握循环队列的入队、出队等操作，学会较为复杂问题的求解。

2）实训内容

n 个人(1,2,3,…,n)围成一圈，从编号为 k 的人开始报数，报数报到 m 的人被淘汰(报数是 1、2、…、m 这样报的)。下次从出队的人之后开始重新报数，循环往复，当队伍中只剩最后一个人的时候，那个人就是胜利者。现在，给定 n、k、m，请你求出胜利者的编号。

相关测试用例如表 3-1 所示。