# 约瑟夫（[Josephus](http://www.baidu.com/link?url=dkcJrPl0Ycwc_jlWbUqBChvWzZdJjpxiStXJv9c2KKqI27SRPeoKTIuFFpH56L3STbXmCXTscs7AZYGS90k6RHqAODVjJqmqAfXQGNaSSRK)）问题

**前言**：约瑟夫问题，有时也称为约瑟夫置换、约瑟夫环、丢手绢问题。

## 一、背景：

39 个犹太人与Josephus及他的朋友躲到一个洞中，39个犹太人决定宁愿死也不要被敌人抓。于是决定了自杀方式，41个人排成一个圆圈，由第1个人开始报数，每报数到第3人该人就必须自杀。然后下一个重新报数，直到所有人都自杀身亡为止。然而Josephus 和他的朋友并不想遵从，Josephus要他的朋友先假装遵从，他将朋友与自己安排在第16个与第31个位置，于是逃过了这场死亡游戏

## 二、拓展：

N个人围成一圈，编号从1到n，约定编号为k（1<=K<=n）的人从1开始报数，数到m的那个人出圈，直到所有人出圈为止，由此产生一个出圈顺序。

如：设N为5，那么就有编号为1、2、3、4、5的5个人，设k=2，m=2，那么首先编号为2（k=2）的人数1，编号为3的人数2，那么编号为3的人就要出圈；然后编号为4的人数1，编号5的人数2，那么编号为5的人出圈；然后编号为1的人数1，编号为2的人数2，那么编号为2的人出圈；然后编号4的人数1，编号1的人数2，那么编号1的人出圈；然后编号为4的人数1，编号为4的人数2，那么编号为4的人出圈；

所以，当N=5，k=2，m=2时最后的出圈顺序为【3、5、2、1、4】

## 三、实现分析：

可以使用环形链表，不论是单项环形链表，还是双向环形链表都可以。

## 四、代码：

### 1、单向链表：

|  |
| --- |
| /\*\*  \* 采用单向环形链表解决约瑟夫问题  \* **@Author**: 陈明 2040820766@qq.com  \* **@Time**: 2019年8月12日 下午5:08:43  \* **@param** nums: 表示总共有多少个人,相当于N  \* **@param** start: 表示从谁开始,相当于k  \* **@param** step: 表示步长,相当于m  \* **@param** returnList : 保存出圈的顺序  \* **@Return**  \*/  **public** **static** **void** JosephusProblem\_1(MyArrayList<Integer> returnList, **int** nums, **int** start, **int** step) {  **if** (nums < 1) {  **return**;  }  **if** (start < 1 || start > nums) {  **return**;  }  // 创建单向环形链表  MySingleCircularLinkedList<Person> list = **new** MySingleCircularLinkedList<Person>();  *createPerson*(list, nums);    //开始报数的人  Person current = list.get(start - 1);  **int** counter = 1;  **while** (list.size() != 0) {  // 表示最终要出圈的人  Person temp = current;  **while** (**true**) {  **if** (counter >= step) {  counter = 1;  current = list.next(temp);  returnList.add(temp.getId());  list.remove(temp);  **break**;  }  temp = list.next(temp);  counter++;  }    }  } |

**【DataStructureAndAlgorithmDemo/src/algorithm/ag\_02\_JosephusProblem/JosephusProblemAlgorithmTest.java】**

|  |
| --- |
|  |

### 2、双向循环链表，顺时针：

|  |
| --- |
| /\*\*  \* 采用双向环形链表解决约瑟夫问题  \*  \* **@Author**: 陈明 2040820766@qq.com  \* **@Time**: 2019年8月12日 下午5:09:19  \* **@Return**  \*/  **public** **static** **void** JosephusProblem\_2(MyArrayList<Integer> returnList, **int** nums, **int** start, **int** step) {    //创建双向循环链表  MyDoubleCircularLinkedList<Person> list = **new** MyDoubleCircularLinkedList<Person>();  *createPerson*(list, nums);    Person current = list.get(start - 1);  **int** counter = 1;  **while** (list.size() != 0) {  Person temp = current;  **while** (**true**) {  **if** (counter >= step) {  counter = 1;  returnList.add(temp.getId());  current = list.next(temp);  list.remove(temp);  **break**;  }  temp = list.next(temp);  counter++;  }    }  } |

**【DataStructureAndAlgorithmDemo/src/algorithm/ag\_02\_JosephusProblem/JosephusProblemAlgorithmTest.java】**

### 3、双向循环链表，逆时针：

|  |
| --- |
| /\*\*  \* 采用双向环形链表解决约瑟夫问题,这里多了一个direction表示是顺时钟数,还是逆时针数  \* 在上面的 JosephusProblem\_2() 方法是采用顺时针的  \* 顺时针时direction=0,逆时针时direction=1  \* **@Author**: 陈明 2040820766@qq.com  \* **@Time**: 2019年8月12日 下午8:49:02  \* **@Return**  \*/  **public** **static** **void** JosephusProblem\_3(MyArrayList<Integer> returnList, **int** nums, **int** start, **int** step, **int** direction) {  **if** (direction == 0) {  *JosephusProblem\_2*(returnList, nums, start, step);  } **else** **if** (direction == 1){  //创建双向循环链表  MyDoubleCircularLinkedList<Person> list = **new** MyDoubleCircularLinkedList<Person>();  *createPerson*(list, nums);    Person current = list.get(start - 1);  **int** counter = 1;  **while** (list.size() != 0) {  Person temp = current;  **while** (**true**) {  **if** (counter >= step) {  counter = 1;  returnList.add(temp.getId());  current = list.prev(temp);  list.remove(temp);  **break**;  }  temp = list.prev(temp);  counter++;  }    }  } **else** {  **throw** **new** RuntimeException("");  }  } |