347 前K个高频元素

Label：Hash
给你一个整数数组 nums 和一个整数 k ，请你返回其中出现频率前 k 高的元素。你可以按 任意顺序 返回答案。
输入: nums = [1,1,1,2,2,3], k = 2
输出: [1,2]

- Hash

```
class Solution {
    public int[] topKFrequent(int[] nums, int k) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i : nums) {
            map.put(i, map.getOrDefault(i, 0) + 1);
        }
        int[] result = new int[k];
        List<Map.Entry<Integer, Integer>> collect =
map.entrySet().stream().sorted(
                (c1, c2) -> c2.getValue().compareTo(c1.getValue())
        ).limit(k).collect(Collectors.toList());

        List<Map.Entry<Integer, Integer>> list = new ArrayList<>(collect);

        for(int i = 0; i < k; i++) {
            result[i] = list.get(i).getKey();
        }
        return result;

    }
}
```

- 流优化

```
class Solution {
    public int[] topKFrequent(int[] nums, int k) {

        Map<Integer, Integer> map = new HashMap<>();
        for (int num : nums) {
            map.put(num, map.getOrDefault(num, 0) + 1);
        }
        return map.entrySet()
                .stream()
                .sorted((m1, m2) -> m2.getValue() - m1.getValue())
                .limit(k)
                .mapToInt(Map.Entry::getKey)
                .toArray();
    }
}
```

- 优先级队列

```
class Solution {
    public int[] topKFrequent(int[] nums, int k) {
        Map<Integer, Integer> occurrences = new HashMap<Integer, Integer>();
        for (int num : nums) {
            occurrences.put(num, occurrences.getOrDefault(num, 0) + 1);
        }

        // int[] 的第一个元素代表数组的值，第二个元素代表了该值出现的次数
        PriorityQueue<int[]> queue = new PriorityQueue<int[]>(new
Comparator<int[]>() {
            public int compare(int[] m, int[] n) {
                return m[1] - n[1];
            }
        });

        for (Map.Entry<Integer, Integer> entry : occurrences.entrySet()) {
            int num = entry.getKey(), count = entry.getValue();
            if (queue.size() == k) { // size == K，就一直维护K
                if (queue.peek()[1] < count) {
                    queue.poll();
                    queue.offer(new int[]{num, count});
                }
            } else {
                queue.offer(new int[]{num, count}); // size < K，就先入队列
            }
        }

        int[] ret = new int[k];
        for (int i = 0; i < k; ++i) {
            ret[i] = queue.poll()[0];  //出队
        }
        return ret;
    }
}
```