一、队列的存储结构

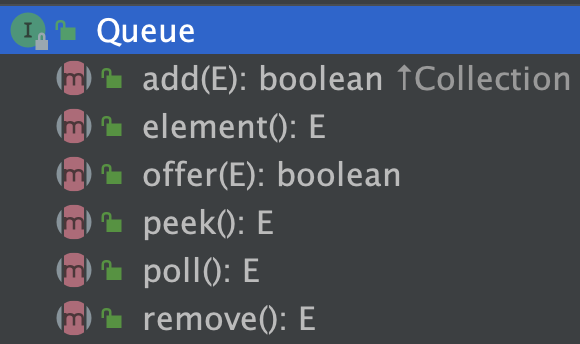
支持先进先出（FIFO）、尾部添加、头部删除

元素从队头读取，从队尾写入。



二、Java Queue接口

Queue接口在java.util包中，并扩展了Collection接口。队列集合用于保存要处理的元素，并提供各种操作。它是一个有序的对象列表。



添加、删除、查询这些个操作都提供了两种形式，其中一种在操作失败时直接抛出异常，而另一种则返回一个特殊的值：



特征：

* Queue用于在队列末尾插入元素并从队列的开头删除。它遵循FIFO概念。
* Java Queue支持Collection接口的所有方法，包括插入，删除等。
* LinkedList，ArrayBlockingQueue、PriorityQueue是最常用的实现。
* 如果对BlockingQueues执行任何null操作，则抛出NullPointerException
* 带有 blocking 的队列是线程安全的，比如 ArrayBlockingQueue
* java.util 包中可用的队列是无界队列 ，使用不当容易造成内存泄漏
* java.util.concurrent 包中可用的队列是有界队列
* 除Deques（双端队列）之外的所有队列分别支持在队列的尾部和头部插入和移除。Deques支撑元件在两端插入和移除

三、Queue接口的重要实现类

#### PriorityQueue

PriorityQueue保存队列元素的顺序不是按加入队列的顺序，而是按队列元素的大小进行重新排序。因此当调用peek()或pool()方法取出队列中头部的元素时，并不是取出最先进入队列的元素，而是取出队列中的最小的元素。

#### PriorityQueue的排序方式

PriorityQueue中的元素可以默认自然排序（也就是数字默认是小的在队列头，字符串则按字典序排列）或者通过提供的Comparator（比较器）在队列实例化时指定的排序方式。关于自然排序与Comparator（比较器）可以参考我在介绍集合Set时的讲解。

注意：队列的头是按指定排序方式的最小元素。如果多个元素都是最小值，则头是其中一个元素——选择方法是任意的。

注意：当PriorityQueue中没有指定Comparator时，加入PriorityQueue的元素必须实现了Comparable接口（即元素是可比较的），否则会导致 ClassCastException。

下面具体写个例子来展示PriorityQueue中的排序方式：

PriorityQueue<Integer> qi = new PriorityQueue<Integer>();

qi.add(5);

qi.add(2);

qi.add(1);

qi.add(10);

qi.add(3);

while (!qi.isEmpty()){

System.out.print(qi.poll() + ",");

}

System.out.println();

//采用降序排列的方式，越小的越排在队尾

Comparator<Integer> cmp = new Comparator<Integer>() {

public int compare(Integer e1, Integer e2) {

return e2 - e1;

}

};

PriorityQueue<Integer> q2 = new PriorityQueue<Integer>(5,cmp);

q2.add(2);

q2.add(8);

q2.add(9);

q2.add(1);

while (!q2.isEmpty()){

System.out.print(q2.poll() + ",");

}