# 概述

* 1. JUC指的是JDK1.5中提供的一套并发包及其子包：java.util.concurrent，java.util.concurrent.lock，java.util.concurrent.atomic
  2. JUC的主要内容：阻塞式队列、并发映射、锁、执行器服务、原子性操作

# BlockingQueue （接口）- 阻塞式队列

### 概述

1. 满足队列的特点：FIFO(First In First Out)
2. BlockingQueue在使用的时候需要容量，且容量是固定的，不可扩容
3. 阻塞：如果队列为空，则试图获取元素的线程会被阻塞；如果队列已满，则试图放入元素的线程会被阻塞
4. BlockingQueue中不允许元素为null
5. 适应于生产消费模型
6. 重要方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 抛出异常 | 返回特殊值 | 永久阻塞 | 定时阻塞 |
| 添加 | add - IllegalStateException | offer - false | put | offer |
| 获取 | remove - NoSuchElementException | poll - null | take | poll |

### 常用方法测试

#### add/remove

##### add：

|  |
| --- |
| public static void main(String[] args) throws InterruptedException {  // 容量指定之后不可变  *BlockingQueue*<String> queue =  new ArrayBlockingQueue<>(5);  // 添加元素  queue.add("a");  queue.add("a");  queue.add("a");  queue.add("a");  queue.add("a");  // 队列已满  // 抛出异常 - IllegalStateException: Queue full  queue.add("b"); } |
|  |

##### remove：

|  |
| --- |
| public static void main(String[] args) throws InterruptedException {   LinkedBlockingQueue<String> queue =  new LinkedBlockingQueue<>();  queue.remove();  } |
|  |

#### offer/poll

##### offer

|  |
| --- |
| public static void main(String[] args) throws InterruptedException {   // 容量指定之后不可变  *BlockingQueue*<String> queue =  new ArrayBlockingQueue<>(5);   // 添加元素  queue.add("a");  queue.add("a");  queue.add("a");  queue.add("a");  queue.add("a");  // 队列满 offer添加返回false  // 返回false  boolean b = queue.offer("c");  System.*out*.println(b);//false } |

##### poll

|  |
| --- |
| public static void main(String[] args) throws InterruptedException {   LinkedBlockingQueue<String> queue =  new LinkedBlockingQueue<>();  // 队列为空 返回null，元素不能为null  System.*out*.println(queue.poll());//null  } |

#### put/take

##### put

|  |
| --- |
| public static void main(String[] args) throws InterruptedException {   // 容量指定之后不可变  *BlockingQueue*<String> queue =  new ArrayBlockingQueue<>(5);  // 添加满元素  queue.add("a");  queue.add("a");  queue.add("a");  queue.add("a");  queue.add("a");  // 队列满产生阻塞  queue.put("d");  System.*out*.println(queue); } |
|  |

##### take

|  |
| --- |
| public static void main(String[] args) throws InterruptedException {  LinkedBlockingQueue<String> queue =  new LinkedBlockingQueue<>();  // 队列没有元素 永久阻塞  System.*out*.println(queue.take()); } |
|  |

#### offer/poll（定时）

|  |
| --- |
| // 定时阻塞 boolean b = queue.offer("e", 5, TimeUnit.*SECONDS*);  // 定时阻塞 System.*out*.println(queue.poll(5, TimeUnit.*SECONDS*)); |

### 常用实现类

#### ArrayBlockingQueue - 阻塞式顺序队列：

* 1. 底层必然基于数组来存储数据
  2. 使用的时候需要指定容量

#### LinkedBlockingQueue - 阻塞式链式队列：

* 1. 底层必然基于节点来存储数据
  2. 在使用的时候可以指定容量也可以不指定。如果指定容量，则容量不可变；如果不指定容量，则容量默认为Integer.MAX\_VALUE = 231-1不可变。因为实际开发中，一般不会在队列中存储21亿个元素，所以一般认为此时的容量是无限的

#### PriorityBlockingQueue - 具有优先级的阻塞式队列：

* 1. 底层是基于数组来存储元素
  2. 使用的时候可以指定容量也可以不指定。如果不指定则默认初始容量是11
  3. PriorityBlockingQueue会对放入的元素来进行排序，要求元素对应的类实现Comparable接口，覆盖compareTo方法指定比较规则。对于不能修改的类，如果要修改排序规则，可在创建PriorityBlockingQueue对象时指定Comparator接口
  4. PriorityBlockingQueue在迭代遍历的时候不保证排序

#### SynchronousQueue - 同步队列

1. 在使用的时候不需要指定容量，容量默认为1且只能为1

### PriorityBlockingQueue - 具有优先级的阻塞式队列

#### 遍历(for i)

输出结果，为排序后

|  |
| --- |
| public static void main(String[] args) throws InterruptedException {   PriorityBlockingQueue<String> queue =  new PriorityBlockingQueue<>();  queue.put("c");  queue.put("a");  queue.put("b");  queue.put("e");  queue.put("f");  queue.put("d");  queue.put("g");  for (int i = 0; i < 5; i++) {  System.*out*.println(queue.take());  } } |
|  |

#### 遍历(for each)

未排序

|  |
| --- |
| public static void main(String[] args) throws InterruptedException {   PriorityBlockingQueue<String> queue =  new PriorityBlockingQueue<>();  queue.put("c");  queue.put("a");  queue.put("b");  queue.put("e");  queue.put("f");  queue.put("d");  queue.put("g");  for (int i = 0; i < 5; i++) {  System.*out*.println(queue.take());  } } |
|  |

#### 排序（实现Comparable接口）

实现compareTo方法

|  |
| --- |
| class Student implements *Comparable*<Student> { |
| // 指定排序规则 // 按照年龄升序 // 升序：this - o // 降序：o - this @Override public int compareTo(Student o) {    return this.age - o.age; } |

#### 排序（自定义Comparator实现类）

当Student中的类不允许更改，这时可以通过*Comparator覆盖*

|  |
| --- |
| // 按照分数降序排序 // 比较器 *Comparator*<Student> c = new Comparator<Student>() {  @Override  public int compare(Student o1, Student o2) {  return o2.getScore() - o1.getScore();  } }; |
| // 在定义队列的时候，传入的比较器的优先级要高于Comparable PriorityBlockingQueue<Student> queue =  new PriorityBlockingQueue<>(5, c); |

# （扩展）BlockingDeque - 阻塞式双端队列

* 1. 特点：允许从两端放入元素也允许从两端获取元素
  2. 遵循阻塞特点，在使用的时候需要指定容量