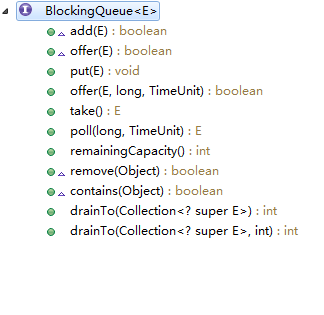
BlockingQueue - 阻塞队列



package java.util.concurrent;

Link:

http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/8-b132/java/util/concurrent/BlockingQueue.java#BlockingQueue

A Interface ->Queue ->Collection ->Iterable

子接口 BlockingDeque

实现类 ArrayBlockingQueue, DelayQueue, LinkedBlockingDeque, LinkedBlockingQueue, PriorityBlockingQueue, SynchronousQueue

A Queue that additionally supports operations

that wait for the queue to become non-empty when retrieving an

element, and wait for space to become available in the queue when

storing an element.

比如经典的“生产者”和“消费者”模型中，通过队列可以很便利地实现两者之间的数据共享。假设我们有若干生产者线程，另外又有若干个消费者线程。如果生产者线程需要把准备好的数据共享给消费者线程，利用队列的方式来传递数据，就可以很方便地解决他们之间的数据共享问题。但如果生产者和消费者在某个时间段内，万一发生数据处理速度不匹配的情况呢？理想情况下，如果生产者产出数据的速度大于消费者消费的速度，并且当生产出来的数据累积到一定程度的时候，那么生产者必须暂停等待一下（阻塞生产者线程），以便等待消费者线程把累积的数据处理完毕，反之亦然。然而，在concurrent包发布以前，在多线程环境下，我们每个程序员都必须去自己控制这些细节，尤其还要兼顾效率和线程安全，而这会给我们的程序带来不小的复杂度。

作为BlockingQueue的使用者，我们再也不需要关心什么时候需要阻塞线程，什么时候需要唤醒线程。

Insert：

add(E):填加元素到队列如果可行的话，并且不阻塞当前线程，成功了返回true，如果没有空间会throw IllegalStateException。（as specified by java.util.Collection.add）

offer(E):同add方法，只是没空间会返回false。（Queue）

put(E):填加元素到队列，如果队列没空间会阻塞当前线程，直到获得可用的空间成功添加。

offer(E，long, TimeUnit):设定等待的时间，如果超时还没成功就返回false。

Remove：

E take()：取走并移除队列Head元素，若为空会阻塞进入等待状态，直到获取到新添加进来的元素。

E poll(long, TimeUnit)：设置取数据等待时间，超时返回null。

int drainTo(Collection）：取出队列所有数据添加到给定的Collection。

Capacity:

int remainingCapacity()：返回在内存或是资源充足情况下队列理论上内容纳的额外值，OR Integer.MAX\_VALUE。

Note:

- A BlockingQueue does not accept {@code null} elements.

- A BlockingQueue may be capacity bounded.

- BlockingQueue implementations are designed to be used primarily for producer-consumer queues,but additionally support Collection interface,so we can use remove(x) when we want to cancel a queued message.

- thread-safe

常用的2种

1.ArrayBlockingQueue

bounded；backed by an array；FIFO

由数组支持的有界阻塞队列，维护一个Object[]，长度在初始化时候赋值，直接进行插入和移除数据操作。队列中的锁没有分离，生产和消费用的是同一个锁。可以内部的线程访问是否是公平锁，默认为非公平锁。

2.LinkedBlockingQueue

optionally-bounded，based on linked nodes；FIFO

由链表支持的阻塞队列，对于对象的操作都是基于Node的，初始化时候可以选择是否设定长度。生产锁和消费锁相互独立，可以并发提高性能。

see demo

reference：

http://www.cnblogs.com/jackyuj/archive/2010/11/24/1886553.html

锁：

http://blog.csdn.net/vking\_wang/article/details/9952063