(/)

Una guía para las colas concurrentes en Java

Última modificación: 19 de agosto de 2020

por Mathieu Fortin (https://www.baeldung.com/author/mathieufortin/)
(https://www.baeldung.com/author/mathieufortin/)

Java (https://www.baeldung.com/category/java/) +
Colecciones de Java (https://www.baeldung.com/category/java/java-collections/)
Cola de Java (https://www.baeldung.com/tag/java-queue/)

1. Información general

En este tutorial, veremos algunas de las principales implementaciones de colas simultáneas en Java. Para obtener una introducción general a las colas, consulte nuestra Guía del (/java-queue) artículo sobre la interfaz de (/java-queue)*cola de* (/java-queue) Java (/java-queue) .

2. Colas

En aplicaciones multiproceso, las colas deben manejar múltiples escenarios productores-consumidores simultáneos. La elección correcta de una cola concurrente podría ser crucial para lograr un buen rendimiento en nuestros algoritmos.

En primer lugar, veremos algunas diferencias importantes entre una cola de bloqueo y una sin bloqueo. Luego, veremos algunas implementaciones y mejores prácticas.

2. Cola con bloqueo frente a sin bloqueo

BlockingQueue (/java-blocking-queue) ofrece un mecanismo simple seguro para subprocesos . En esta cola, los subprocesos deben esperar la disponibilidad de la cola. Los productores esperarán la capacidad disponible antes de agregar elementos, mientras que los consumidores esperarán hasta que la cola esté vacía. En esos casos, la cola sin bloqueo lanzará una excepción o devolverá un valor especial, como *nulo* o *falso* .

Para lograr este mecanismo de bloqueo, la interfaz *BlockingQueue* expone dos funciones además de las funciones normales de *Cola* : *poner* y *tomar* . Esas funciones equivalen a *agregar* y *eliminar* en una *cola* estándar .

3. Implementaciones de *colas* simultáneas

3.1. *ArrayBlockingQueue*

Como sugiere su nombre, esta cola usa una matriz internamente. Como consecuencia, es una cola limitada, lo que significa que tiene un tamaño fijo .

Una cola de trabajo simple es un caso de uso de ejemplo. Este escenario suele ser una proporción baja de productor a consumidor, en la que dividimos las tareas que requieren mucho tiempo entre varios trabajadores. Dado que esta cola no puede crecer indefinidamente, el límite de tamaño actúa como un umbral de seguridad si la memoria es un problema .

Hablando de memoria, es importante tener en cuenta que la cola preasigna la matriz. Si bien esto puede mejorar el rendimiento, también puede consumir más memoria de la necesaria . Por ejemplo, una cola de gran capacidad puede permanecer vacía durante largos períodos de tiempo.

Además, el *ArrayBlockingQueue* utiliza un candado para ambos *de venta* y *para llevar* las operaciones. Esto asegura que no se sobrescriban las entradas, a costa de un impacto en el rendimiento.

3.2. *LinkedBlockingQueue*

El *LinkedBlockingQueue* (/java-queue-linkedblocking-concurrentlinked#linkedblockingqueue) utiliza un *LinkedList* (/java-linkedlist) variante, donde cada elemento de cola es un nuevo nodo. Si bien esto hace que la cola sea ilimitada en principio, todavía tiene un límite *estricto* de *Integer.MAX_VALUE* .

Por otro lado, podemos establecer el tamaño de la cola usando el constructor *LinkedBlockingQueue (capacidad int)* .

Esta cola utiliza bloqueos distintos para *poner* y *tomar* las operaciones. Como consecuencia, ambas operaciones se pueden realizar en paralelo y mejorar el rendimiento.

Dado que *LinkedBlockingQueue* puede ser delimitado o ilimitado, ¿por qué *usaríamos ArrayBlockingQueue* sobre este? *LinkedBlockingQueue* necesita asignar y desasignar nodos cada vez que se agrega o quita un elemento de la cola . Por esta razón, *ArrayBlockingQueue* puede ser una mejor alternativa si la cola crece y se reduce rápidamente.

Se dice que el rendimiento de *LinkedBlockingQueue* es impredecible. En otras palabras, siempre necesitamos perfilar nuestros escenarios para asegurarnos de que usamos la estructura de datos correcta.

3.3. *PriorityBlockingQueue*

El *PriorityBlockingQueue* (/java-priority-blocking-queue) es nuestro ir a la solución cuando tenemos que consumir artículos en un orden específico . Para lograr esto, *PriorityBlockingQueue* usa un montón binario basado en arreglos.

Si bien internamente utiliza un mecanismo de bloqueo único, la operación de *toma* puede ocurrir simultáneamente con la operación de *colocación* . El uso de un simple spinlock lo hace posible.

Un caso de uso típico es consumir tareas con diferentes prioridades. No queremos que una tarea de baja prioridad sustituya a una de alta prioridad .

3.4. *DelayQueue*

Usamos *DelayQueue* (/java-delay-queue) cuando un consumidor solo puede tomar un artículo vencido . Curiosamente, utiliza una *PriorityQueue* internamente para ordenar los artículos por su vencimiento.

Dado que esta no es una cola de uso general, no cubre tantos escenarios como *ArrayBlockingQueue* o *LinkedBlockingQueue* . Por ejemplo, podemos usar esta cola para implementar un ciclo de eventos simple similar al que se encuentra en NodeJS. Colocamos las tareas asincrónicas en la cola para su posterior procesamiento cuando vencen.

3.5. *LinkedTransferQueue*

El *LinkedTransferQueue* (/java-transfer-queue) introduce una *transferencia* método. Mientras que otras colas normalmente se bloquean al producir o consumir artículos, *LinkedTransferQueue* permite al productor esperar el consumo de un artículo .

Usamos *LinkedTransferQueue* cuando necesitamos una garantía de que alguien ha tomado un artículo en particular que ponemos en la cola. Además, podemos implementar un algoritmo de contrapresión simple usando esta cola. De hecho, al bloquear a los productores hasta el consumo, los consumidores pueden impulsar el flujo de mensajes producidos.

3.6. *SynchronousQueue*

Si bien las colas suelen contener muchos elementos, *SynchronousQueue* (/java-synchronous-queue) siempre tendrá, como máximo, un solo elemento. En otras palabras, necesitamos ver *SynchronousQueue* como una forma sencilla de intercambiar algunos datos entre dos hilos .

Cuando tenemos dos subprocesos que necesitan acceso a un estado compartido, a menudo los sincronizamos con *CountDownLatch* (/java-countdown-latch) u otros mecanismos de sincronización. Al usar *SynchronousQueue* , podemos evitar esta sincronización manual de subprocesos .

3.7. *ConcurrentLinkedQueue*

El *ConcurrentLinkedQueue* (/java-queue-linkedblocking-concurrentlinked#concurrentlinkedqueue) es la única cola de no bloqueo de esta guía. En consecuencia, proporciona un algoritmo "sin espera" en el que se garantiza que *agregar* y *sondear* son seguros para subprocesos y que regresan de inmediato . En lugar de bloqueos, esta cola utiliza CAS (Compare-And-Swap) (/lock-free-programming) .

Internamente, se basa en un algoritmo de algoritmos de *cola simultánea de bloqueo y no bloqueo simples, rápidos y prácticos* (https://www.cs.rochester.edu/u/scott/papers/1996_PODC_queues.pdf) de Maged M. Michael y Michael L. Scott.

Es un candidato perfecto para los sistemas reactivos modernos (/java-reactive-systems) , donde el uso de estructuras de datos de bloqueo a menudo está prohibido.

Por otro lado, si nuestro consumidor termina esperando en un bucle, probablemente deberíamos elegir una cola de bloqueo como una mejor alternativa.

4. Conclusión

En esta guía, analizamos diferentes implementaciones de colas simultáneas, discutiendo sus fortalezas y debilidades. Teniendo esto en cuenta, estamos mejor equipados para desarrollar sistemas eficientes, duraderos y disponibles.

0 COMENTARIOS

CATEGORÍAS

PRIMAVERA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
DESCANSO (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SEGURIDAD (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCIA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
LADO DEL CLIENTE HTTP (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIE

TUTORIAL DE JAVA "VOLVER A LO BÁSICO" (/JAVA-TUTORIAL)
TUTORIAL DE JACKSON JSON (/JACKSON)
TUTORIAL DE HTTPCLIENT 4 (/HTTPCLIENT-GUIDE)
DESCANSO CON SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
TUTORIAL DE PERSISTENCIA DE PRIMAVERA (/PERSISTENCE-WITH-SPRING-SERIES)
SEGURIDAD CON SPRING (/SECURITY-SPRING)

ACERCA DE

SOBRE BAELDUNG (/ABOUT)
LOS CURSOS (HTTPS://COURSES.BAELDUNG.COM)
TRABAJOS (/TAG/ACTIVE-JOB/)
EL ARCHIVO COMPLETO (/FULL_ARCHIVE)
ESCRIBE PARA BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORES (/EDITORS)
NUESTROS COMPAÑEROS (/PARTNERS)