Bibliotekos java.util.concurrent galimybės

Laimonas Beniušis
Kompiuterių Mokslas 1g

Užraktai

java.util.concurrent.locks

ReentrantLock

Ta pati gija gali užrakinti kelis kartus be “deadlock” situacijos

ReentrantReadWriteLock

Užraktas su skirtingais užrakinimo tipais (read/write)

Neblokuojantis algoritmas

Compare And Swap (CAS)

Inkrementacijos pavyzdys turint tik CAS funkciją.

```
do{
    int current = this.get();
    int next = current + 1;
}while(!this.compareAndSet(current, next));
```

“Bandyk tol, kol nebus kolizijos”

“Atominiai” kintamieji

java.util.concurrent.atomic

Skaitymo veikimu nesiskiria nuo *volatile*, rašymas vyksta atomiškai, t.y. operacijos “nuskaityk” ir “įrašyk” yra sujungiamos.

Veikimas ir implementacija priklauso nuo architektūros, t.y. jeigu nėra tokios procesoriaus instrukcijos naudojamas CAS arba užraktai.

Įvairūs įrankiai

CountDownLatch

Blokuoja tol, kol yra vidinė reikšmė pasiekia 0.

Negalima reset’inti.

CyclicBarrier

Blokuoja tol, kol tam tikras kiekis gijų pasiekia barjerą. Tada paleidžia visas pasiekusias. Gijų kiekis, nėra dinamiškas.

Įvairūs įrankiai

Exchanger

2 gijos gali apsikeisti elementais. Blokuojama abiejose gijose, iki tol, kol mainai pavyksta.

Phaser

“Dinamiškas” *CyclicBarrier*.

Suskirsto darbą į “fazes”. Fazėse dalyvaujančių gijų kiekis gali kisti.

Collections

CopyOnWriteArrayList

CopyOnWriteArraySet

Kopijuoja visus elementus, kai įvyskta pokytis.

Gali būti naudinga, kada blokuojantys algoritmai per lėti ir pokyčiai nutinka retai (palyginus su skaitymais).

ConcurrentHashMap ConcurrentNavigableMap

Pavadinimas	HashMap	SynchronizedMap*	HashTable**	ConcurrentHashMap
Null key	Taip		Ne	
Thread-safe	Ne	Taip		
Užrakto sritis	Nėra	Visas		Dalis
Iterator	Fail-fast			Fail-safe

*gaunamas iš Collections.synchronizedMap(Map map)
**pasenęs (deprecated)

BlockingQueue ir BlockingDeque

BlockingQueue

	Throws exception	*Special value*	*Blocks*	*Times out*
Insert	`add(e)`	`offer(e)`	`put(e)`	`offer(e, time, unit)`
Remove	`remove()`	`poll()`	`take()`	`poll(time, unit)`
Examine	`element()`	`peek()`	*not applicable*	*not applicable*

Implementacijos:

- *ArrayBlockingQueue*
- *DelayQueue*
- *LinkedBlockingQueue*
- *LinkedTransferQueue*
- *PriorityBlockingQueue*
- *SynchronousQueue*

BlockingDeque

First Element (Head)				
	Throws exception	*Special value*	*Blocks*	*Times out*
Insert	`addFirst(e)`	`offerFirst(e)`	`putFirst(e)`	`offerFirst(e, time, unit)`
Remove	`removeFirst()`	`pollFirst()`	`takeFirst()`	`pollFirst(time, unit)`
Examine	`getFirst()`	`peekFirst()`	*not applicable*	*not applicable*
Last Element (Tail)				
	Throws exception	*Special value*	*Blocks*	*Times out*
Insert	`addLast(e)`	`offerLast(e)`	`putLast(e)`	`offerLast(e, time, unit)`
Remove	`removeLast()`	`pollLast()`	`takeLast()`	`pollLast(time, unit)`
Examine	`getLast()`	`peekLast()`	*not applicable*	*not applicable*

Implementacijos:

- *LinkedBlockingDeque*

Įdomios išimtys

DelayQueue

Savotiško rūšiavimo *PriorityBlockingQueue*

Elementai turi būti implementuoti *Delayed* interface’ą.

Elementai rūšiuojami pagal likusį laiką.

Įdomios išimtys

SynchronousQueue

Nelabai galima vadinti eile, labiau susitikimo taškas.

Elementai gali būti pridedami tik tada, kada yra kam juos pasiimti.

„Prekė pristatoma po užsakymo“

Įdomios išimtys

TransferQueue

Gali veikti kaip normalus *Collection*, tačiau turi metodus: *tryTransfer* ir *transfer*, kurie veikia panašiai kaip *SynchronousQueue*. Skiriasi tik tuo, kad laukia kol elementas bus pasiimtas.

„Prekė pristatoma prieš užsakymą“

Callable

Callable interface'as, naudojamas panašiai kaip *Runnable*, tačiau **grąžiną reikšmę** ir gali mesti išimtis (throws Exception)

Callable neturi *run* metodo, todėl reikia gaubiančių klasių, norint sukurti *Thread* pvz. *FutureTask.*

Callable (*Runnable* irgi) gali būti naudojamas kartu su vienu iš daugelio *Executor* implementacijų.

FutureTask ir *CompletableFuture*

Turi vidines būsenas, todėl negali būti paprastai paleidžiama antrą kartą (tik specialiu metodu *runAndReset*).

CompletableFuture naudojamas įgyvendinti asinchroniška “Reactive” veiklą (vienas įvykis lemia kitą).

Executors ir ThreadPools

ExecutorService:

- *AbstractExecutorService* (daug įvairių)
- *ForkJoinPool*
- *ThreadPoolExecutor*

ScheduledExecutorService:

- ScheduledThreadPoolExecutor

Įdomios išimtys

ScheduledThreadPoolExecutor

Galima daryti užduotį intervalais, arba užsakyti užduotį po tam tikro laiko tarpo.

Jeigu ansktesnė užduotis nebuvo baigta, ja gali užsiimti laisva gija.

Įdomios išimtys

ForkJoinPool

Naudoja *RecursiveTask* (kaip *Callable*) arba *RecursiveAction* (kaip *Runnable*)

Viduje yra metodai *fork* ir *join* kurie simuliuoja užduoties rekursinį padalinimą į naujas gijas, kuriomis užsiima gija paskirtas *ForkJoinPool*.

Pabaiga?


YO DAWG, I HEARD YOU LIKE MULTITHREADING
SO WE MADE THREADS CALLING OTHER THREADS SO YOU CAN MULTITHREAD WHILE YOU MULTITHREAD
quickmeme.com