Fork-Join framework

Fork-Join framework

- Fork – vilka, bo’lsih
- Join - kutish

Fork-Join framework 1

- The fork-join framework allows to break a certain task on several workers and then wait for the result to combine them.
- It leverages multi-processor machine's capacity to great extent.
- Fork-join FrameWork ishni birnechta vazifalarga bo’lib bajarish va natijasini birlashtirish imkonini beradi.
- U Kompyuterni barcha protsessorlaridan to’liq foydalanish imkonini beradi.

Fork-Join framework 2

- The fork/join framework was presented in Java 7.
- It provides tools to help speed up parallel processing by attempting to use all available processor cores – which is accomplished **through a divide and conquer approach**.
- In practice, this means that **the framework first "forks"**, recursively breaking the task into smaller independent subtasks until they are simple enough to be executed asynchronously.
- After that, **the "join" part begins**, in which results of all subtasks are recursively joined into a single result, or in the case of a task which returns void, the program simply waits until every subtask is executed.
- Fork - join framework java 7 da tagdim etilgan. Bu shunday mexanizmki u mavjut bo'lgan barcha protsessorlarni ishlatib parallel dasturlash ni tezligini oshirish uchun ishlatiladi, bo'l va zabt et yondashuv orqali amalga oshiriladi.
- Amalda **framework ishni ansinxron ravishda bajarish uchun etarlicha sodda/kichik bo'lgunga qadar kichik ishlarga recursive tarzda bo'lib tashlaydi.**
- Shundan keyin 'join' qismi boshlanadi. Bunda pastgi Thread larning javoblari rekursiv tarzda bitta jaobga yig'iladi. Agar task void qaytarsa, dastur barcha subTask larni tugashini kutadi.

dasturlash.uz

Fork-Join framework 2

- To provide effective parallel execution, the fork/join framework uses a pool of threads called the *ForkJoinPool*, which manages worker threads of type *ForkJoinWorkerThread*.
- Fork-join framework ki samarali parallel ishlashni tanimlash uchun *ForkJoinPool* deb nomlangan Thread lar poolini ishlatadi. ForskJoinPool classi ForkJoinWorkerThread tipidagi ishchi Thread larni boshqaradi.

ForkJoinPool

- The *ForkJoinPool* is the heart of the framework.
- It is an implementation of the *ExecutorService* that manages worker threads and provides us with tools to get information about the thread pool state and performance.
- Worker threads can execute only one task at a time.
- *ForkJoinPool* doesn't create a separate thread for every single subtask. Instead, each thread in the pool has its own double-ended queue (or deque, pronounced deck) which stores tasks.
- *ForkJoinPool* framework ning yuragi hisoblanadi.
- U ExecutorService dan implements olgan va ishchi Thread larni boshqaradi, bundan tashqari Thread pool ni ishlashi va holati haqida malumot olish uchun methodlar taqdim etadi.
- Worker Thread lar birvaqtni o'zida bitta ishni bajaradi.
- ForkJoinPoolhar har bitta subTask ga alohida Thread yaratmaydi. Pool dagi har bitta Thread ni o'zini vazifani saqlaydigan navbati bo'ladi. dasturlash.uz

ForkJoinTask<V>

- *ForkJoinTask* is the base type for tasks executed inside *ForkJoinPool*.
- In practice, one of its two subclasses should be extended: the *RecursiveAction* for *void* tasks and the *RecursiveTask<V>* for tasks that return a value. They both have an abstract method *compute()* in which the task's logic is defined.

- ForkJoinTask - bu ForkJoinPool ichida bajariladigan vazifalarning asosiy turi.
- Amalda uning ikkita subClassidan nasil olish kerak: RecursiveAction classi natija qaytarmaydigan ishlar uchun va RecursiveTask<V> classi natija qaytaradigan vazifalar uchun. Ikkala class da ham fazifani yozish uchun compute() degan abstract method bor.

ForkJoinTask - RecursiveAction

- RecursiveAction represents a task which does not return any value.

```
class Writer extends RecursiveAction {
   @Override
   protected void compute() { }
}
```

ForkJoinTask - RecursiveTask

- RecursiveTask represents a task which returns a value.

```
class Sum extends RecursiveTask<Long> {
   @Override
   protected Long compute() { return null; }
}
```

ForkJoinPool

ForkJoinTask Methods

- fork(), which allows a ForkJoinTask to be scheduled for asynchronous execution (launching a new subtask from an existing one).
  - Bu method ForkJoinTask ni asinxron ravishda ishlashini taminlaydi. (hozirgi taskdan subTask yaratadi.)
- join(), which returns the result of the computation when it is done, allowing a task to wait for the completion of another one.
  - Hozirgi task boshqa taskni (subTaskni) kutib turishi ni taminlaydi va undan javobni qaytaradi.
- compute(), methodi ishni reqursiv ravishda ishga tushuradi.

ForkJoinTask Methods Explanation 1

- First, you have to decide when the problem is small enough to solve directly. This acts as the base case. A big task is divided into smaller tasks recursively until the base case is reached.
- Each time a task is divided, you call the fork() method to place the first subtask in the current thread's deque, and then you call the compute() method on the second subtask to recursively process it.
- Birinchi ish hisoblash (hal qilish) uchun yetarli darajada kichik bo'lguncha bo'laklarga bo'lish kerak. Bu asosiy holat dir. Katta ish kichik ishlarga recursive tarzda bo'linadi.
- Har gal task 2ga bo'linganda u 2ta subTask ga bo'linadi. Uning birinchi subTask ni hozirgi Thread ni dequega (navbatiga) qo'yish uchun fork() methodini chaqiramiz. Ikkinchi subTask ni reqursive bajarish uchun compute() methodini chaqiramiz.

ForkJoinTask Methods Explanation 2

- Finally, to get the result of the first subtask you call the join() method on this first subtask. This should be the last step because join() will block the next program from being processed until until the result is returned.
- fork() before join(), there won't be any result to retrieve. If you call join() before compute(), the program will perform like if it was executed in one thread and you'll be wasting time.
- If you follow the right order, while the second subtask is recursively calculating the value, the first one can be stolen by another thread to process it. This way, when join() is finally called, either the result is ready or you don't have to wait a long time to get it.

- Nihoyat birinchi subTask ni natijasini olish uchun birinchi subTask uchun join() methodni chaqiramiz. Bu ohirgi bosqishda bo’lishi kerak, sababi join() methodi hozirgi threadni toki javob kelmagunicha blocklaydi.
- fork() dan oldin joint() methodni chaqirsak hech qanday javob qaymaydi. Agar siz join() ni compute() dan oldin ishlatsangiz bu dastur xuddi thread da ishlagandek bo’ladi va siz vaqt yo’qotasiz.
- Agar siz to'g'ri tartibda bajarsangiz, Ikkinchi task recursiv tarzda bajarilayotgnda Birinchi subTask boshqa Thread tomonidan navbatdan olinib ishlatilishi mumkin. Shu tarzda birinchi subTask uchun join() chaqirilganda, javob alla qachon tayyor bo’ladi yoki siz uni ko’p kutib turmaysiz.

Fork-Join framework Example 1

```
static class Sum extends RecursiveTask<Long> {
    int low; int high; int[] array;
    Sum(int[] array, int low, int high) { ….. }

    protected Long compute() {
        if(high - low <= 10) {
            long sum = 0;
            for(int i = low; i < high; ++i)
                sum += array[i];
            return sum;
        } else {
            int mid = low + (high - low) / 2;
            Sum left  = new Sum(array, low, mid);
            Sum right = new Sum(array, mid, high);
            left.fork();
            long rightResult = right.compute();
            long leftResult  = left.join();
            return leftResult + rightResult;
        }
    }
}
```

- RecursiveTask Clas

Fork-Join framework Example 2

- Main

```
public static void main(String[] args) {
        int nThreads = Runtime.getRuntime().availableProcessors();

        System.out.println(nThreads);

        int[] numbers = new int[1000];

        for(int i = 0; i < numbers.length; i++) {
            numbers[i] = i;
        }

        ForkJoinPool forkJoinPool = new ForkJoinPool(nThreads);
        Long result = forkJoinPool.invoke(new Sum(numbers,0,numbers.length));
        System.out.println(result);

    }
```

ForJoinFrameWork Links

- https://www.baeldung.com/java-fork-join
- https://www.pluralsight.com/guides/introduction-to-the-fork-join-framework
- https://www.tutorialspoint.com/java_concurrency/concurrency_fork_join.htm