The Java Fork-Join Pool Framework

(Part 6)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Understand how the Java fork-join framework processes tasks in parallel
- Recognize the structure & functionality of the fork-join framework
- Know how the fork-join framework is implemented internally
- Recognize the key methods in the ForkJoinPool class & related classes
- Apply the fork-join framework in practice
- Be aware of the common fork-join pool
Overview of the Common Fork-Join Pool
Overview of the Common Fork-Join Pool

A static common pool is available & appropriate for most programs.

```java
public static ForkJoinPool commonPool()

Returns the common pool instance. This pool is statically constructed; its run state is unaffected by attempts to shutdown() or shutdownNow(). However this pool and any ongoing processing are automatically terminated upon program System.exit(int). Any program that relies on asynchronous task processing to complete before program termination should invoke commonPool().awaitQuiescence, before exit.

Returns:
the common pool instance

Since:
1.8
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html#commonPool
Overview of the Common Fork-Join Pool

- A static common pool is available & appropriate for most programs
- The common pool is used by any ForkJoinTask that is not explicitly submitted to a specified pool

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html#commonPool
Overview of the Common Fork-Join Pool

- A static common pool is available & appropriate for most programs
  - The common pool is used by any ForkJoinTask that is not explicitly submitted to a specified pool
  - The common pool may optimize resource utilization since it’s aware what cores are being used globally
Overview of the Common Fork-Join Pool

- A static common pool is available & appropriate for most programs
  - The common pool is used by any ForkJoinTask that is not explicitly submitted to a specified pool
  - The common pool may optimize resource utilization since it’s aware what cores are being used globally
    - This “global” vs “local” resource management tradeoff is common in computing & other domains

See blog.tsia.com/blog/local-or-global-resource-management-which-model-is-better
Overview of the Common Fork-Join Pool

- A static common pool is available & appropriate for most programs
  - The common pool is used by any ForkJoinTask that is not explicitly submitted to a specified pool
  - The common pool may optimize resource utilization since it’s aware what cores are being used globally
- This pool is also used by the Java 8 parallel streams framework

See dzone.com/articles/common-fork-join-pool-and-streams
Overview of the Common Fork-Join Pool

By default the common ForkJoinPool has one less thread than the number of cores:

```java
ForkJoinPool makeCommonPool() {
    ...
    parallelism = Runtime.getRuntime().availableProcessors() - 1;
    ...
}
```

E.g., returns 8 on my quad-core hyper-threaded processor.

See docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html#availableProcessors
Overview of the Common Fork-Join Pool

- By default the common ForkJoinPool has one less thread than the # of cores

```java
ForkJoinPool makeCommonPool() {
    ...
    parallelism = Runtime.getRuntime().availableProcessors() - 1;
    ...
}
```

e.g., returns 7 on my quad-core hyper-threaded processor

```java
System.out.println("The parallelism in the" + "common fork-join pool is " + ForkJoinPool
                     .getCommonPoolParallelism());
```

Overview of the Common Fork-Join Pool

- By default the common ForkJoinPool has one less thread than the number of cores.

A Java program can leverage all cores since it uses the invoking thread, e.g., main thread.
Overview of the Common Fork-Join Pool

- However, the default # of threads in the fork-join pool may be inadequate
Overview of the Common Fork-Join Pool

- However, the default # of threads in the fork-join pool may be inadequate
- e.g., problems occur when blocking operations are used in a parallel stream

These problems may range from underutilization of processor cores to deadlock..

E.g., downloading more images than # of cores
Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically
The common pool size can thus be expanded & contracted programmatically

By modifying a system property

```java
int numberOfThreads = 10;
System.setProperty("java.util.concurrent.ForkJoinPool.common.parallelism",
                    numberOfThreads);
```

Overview of the Common Fork-Join Pool

It's hard to estimate the total # of threads to set in the common fork-join pool
Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically by modifying a system property.

```java
int numberOfThreads = 10;
System.setProperty("java.util.concurrent.ForkJoinPool.common.parallelism", numberOfThreads);
```

*Modifying this property affects all common fork-join usage in a process!"
Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically
- By modifying a system property

```java
int numberOfThreads = 10;
System.setProperty("java.util.concurrent.ForkJoinPool.common.parallelism", 
"numberofThreads");
```

It’s thus necessary to be able to automatically increasing fork/join pool size
Overview of the Common Fork-Join Pool

• The common pool size can thus be expanded & contracted programmatically
  • By modifying a system property
  • By using a ManagedBlocker

### Interface ForkJoinPool.ManagedBlocker

Enclosing class:
ForkJoinPool

```java
public static interface ForkJoinPool.ManagedBlocker
```

Interface for extending managed parallelism for tasks running in ForkJoinPools.

A ManagedBlocker provides two methods. Method `isReleasable()` must return `true` if blocking is not necessary. Method `block()` blocks the current thread if necessary (perhaps internally invoking `isReleasable` before actually blocking). These actions are performed by any thread invoking `ForkJoinPool.managedBlock(ManagedBlocker)`. The unusual methods in this API accommodate synchronizers that may, but don't usually, block for long periods. Similarly, they allow more efficient internal handling of cases in which additional workers may be, but usually are not, needed to ensure sufficient parallelism. Toward this end, implementations of method `isReleasable` must be amenable to repeated invocation.

Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically
  - By modifying a system property
  - By using a ManagedBlocker
  - Temporarily add worker threads to the common fork-join pool
The common pool size can thus be expanded & contracted programmatically

- By modifying a system property
- By using a ManagedBlocker
  - Temporarily add worker threads to the common fork-join pool
- Useful when behaviors block on I/O and/or synchronizers

ManageBlockers can only be used with the common fork-join pool.
Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically
  - By modifying a system property
  - By using a ManagedBlocker
    - Temporarily add worker threads to the common fork-join pool
  - Useful when behaviors block on I/O and/or synchronizers

```java
SupplierManagedBlocker<T> mb = new SupplierManagedBlocker<>(supplier);
...
ForkJoinPool.managedBlock(mb);
return mb.getResult();
```

See Part 7 of this lesson for information on the ManageBlocker interface
Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically
  - By modifying a system property
  - By using a ManagedBlocker
    - Temporarily add worker threads to the common fork-join pool
    - Useful when behaviors block on I/O and/or synchronizers
  - ForkJoinPool reclaims threads during periods of non-use & reinstates them on later use

*A pool of worker threads*
End of the Java Fork-Join Pool Framework (Part 6)