The Java Fork-Join Pool Framework

(Part 7)

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
- Be able to use ManagedBlocker to avoid starvation & improve performance

Interface ForkJoinPool.ManagedBlocker

Enclosing class:
ForkJoinPool

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.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.ManagedBlocker.html
Overview of the ManagedBlocker Interface
Overview of the ManagedBlocker Interface

• The Java fork-join framework is largely designed for tasks that “run to completion” without blocking

See en.wikipedia.org/wiki/Run_to_completion_scheduling
Overview of the ManagedBlocker Interface

ManagedBlocker handles cases where more worker threads may be needed to ensure liveness/responsiveness for blocking operations.

Overview of the ManagedBlocker Interface

- ManagedBlocker handles cases where more worker threads may be needed to ensure liveness/responsiveness for blocking operations
  - e.g., to automatically/temporarily increase common fork/join pool size
Overview of the ManagedBlocker Interface

• ForkJoinPool reclaims threads during periods of non-use & reinstates them on later use
Overview of the ManagedBlocker Interface

- ForkJoinPool reclaims threads during periods of non-use & reinstates them on later use
- It also tries to create or activate threads to ensure the target level of parallelism is met
Overview of the ManagedBlocker Interface

- ManagedBlocker defines two methods

```java
interface ManagedBlocker {
    boolean isReleasable();

    boolean block();
}
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.ManagedBlocker.html
Overview of the ManagedBlocker Interface

- ManagedBlocker defines two methods
- Returns true if blocking is unnecessary

```java
interface ManagedBlocker {
    boolean isReleasable();
    boolean block();
}
```

e.g., was able to acquire a lock or a message without blocking
Overview of the ManagedBlocker Interface

- ManagedBlocker defines two methods
  - Returns true if blocking is unnecessary
  - Possibly blocks the calling thread

```java
interface ManagedBlocker {
    boolean isReleasable();
    boolean block();
}
```

e.g., waiting for a lock or I/O operation
Overview of the ManagedBlocker Interface

- ManagedBlocker defines two methods
  - Returns true if blocking is unnecessary
  - Possibly blocks the calling thread
  - Returns true if no additional blocking is necessary

```java
interface ManagedBlocker {
    boolean isReleasable();
    boolean block();
}
```

*i.e., if isReleasable() would return true*
How the Java Fork-Join Pool Applies ManagedBlocker
How the Java Fork-Join Pool Applies ManagedBlocker

- The ForkJoinPool class uses a ManagedBlocker internally

```java
class ForkJoinPool extends AbstractExecutorService {
    ...
    static void managedBlock(ManagedBlocker blocker) {
        ...
        while (!blocker.isReleasable()) {
            if (p.tryCompensate(p.ctl)) {
                ...
                do {} // This loop continues until blocker becomes releasable
                while (!blocker.isReleasable() && !blocker.block());
            }
            ...
        }
    }
    ...
}
```

See [openjdk/7-b147/java/util/concurrent/ForkJoinPool.java](openjdk/7-b147/java/util/concurrent/ForkJoinPool.java)
How the Java Fork-Join Pool Applies ManagedBlocker

- The ForkJoinPool class uses a ManagedBlocker internally

```java
class ForkJoinPool extends AbstractExecutorService {
    ...
    static void managedBlock(ManagedBlocker blocker) {
        ...
        while (!blocker.isReleasable()) {
            if (p.tryCompensate(p.ctl)) {
                ...
                do {} 
                while (!blocker.isReleasable())
                && !blocker.block());
            ...
        }
        ...
    }
    ...
}
```

This method activates a spare thread to ensure sufficient parallelism while calling thread is blocked.

See openjdk/7-b147/java/util/concurrent/ForkJoinPool.java
How the Java Fork-Join Pool Applies ManagedBlocker

- The ForkJoinPool class uses a ManagedBlocker internally

```java
class ForkJoinPool extends AbstractExecutorService {
    ...
    static void managedBlock(ManagedBlocker blocker) {
        ...
        while (!blocker.isReleasable()) {
            if (p.tryCompensate(p.ctl)) {
                ...
                do {}
                while (!blocker.isReleasable() && !blocker.block());
            }
        }
    }
    ...
}
```

If there aren’t enough live threads, create or re-activate a spare thread to compensate for blocked joiners ‘til they unblock.
The ForkJoinPool class uses a ManagedBlocker internally.

class ForkJoinPool extends AbstractExecutorService {
  ...
  static void managedBlock(ManagedBlocker blocker) {
    ...
    while (!blocker.isReleasable()) {
      if (p.tryCompensate(p.ctl)) {
        ...
        do {} 
        while (!blocker.isReleasable())
          && !blocker.block();
      }
    ...
  }
  ...

  Potentially block the calling thread
Applying the Managed Blocker Interface
Applying the ManagedBlocker Interface

- This example applies a ManagedBlocker on a ReentrantLock (from Java docs)

```java
class ManagedLocker implements ManagedBlocker {
    final ReentrantLock mLock;
    boolean mHasLock = false;

    ManagedLocker(ReentrantLock lock) { mLock = lock; }

    public boolean isReleasable() {
        return mHasLock || (mHasLock = mLock.tryLock());
    }

    public boolean block() {
        if (!mHasLock) {
            mLock.lock();
        }
        return true;
    }
}
```

Handles a blocking synchronizer
Applying the ManagedBlocker Interface

- This example applies a ManagedBlocker on a ReentrantLock (from Java docs)

```java
public class ManagedLocker implements ManagedBlocker {
    final ReentrantLock mLock;
    boolean mHasLock = false;

    ManagedLocker(ReentrantLock lock) {
        mLock = lock;
    }

    public boolean isReleasable() {
        return mHasLock || (mHasLock = mLock.tryLock());
    }

    public boolean block() {
        if (!mHasLock) {
            mLock.lock();
        }
        return true;
    }
}
```

Constructor stores the lock
Applying the ManagedBlocker Interface

- This example applies a ManagedBlocker on a ReentrantLock (from Java docs)

```java
class ManagedLocker implements ManagedBlocker {
    final ReentrantLock mLock;
    boolean mHasLock = false;

    ManagedLocker(ReentrantLock lock) { mLock = lock; }

    public boolean isReleasable() {
        return mHasLock || (mHasLock = mLock.tryLock());
    }

    public boolean block() {
        if (!mHasLock)
            mLock.lock();
        return true;
    }
}
```

Tries to acquire the lock (non-blocking)
Applying the ManagedBlocker Interface

• This example applies a ManagedBlocker on a ReentrantLock (from Java docs)

class ManagedLocker implements ManagedBlocker {
    final ReentrantLock mLock;
    boolean mHasLock = false;

    ManagedLocker(ReentrantLock lock) { mLock = lock; }

    public boolean isReleasable() {
        return mHasLock || (mHasLock = mLock.tryLock());
    }

    public boolean block() {
        if (!mHasLock)
            mLock.lock();
        return true;
    }
}
Applying the ManagedBlocker Interface

- This example applies a ManagedBlocker on a BlockingQueue (from Java docs)

```java
class QueueTaker<E> implements ManagedBlocker {
    final BlockingQueue<E> mQueue;
    volatile E mItem = null;

    QueueTaker(BlockingQueue<E> q) { mQueue = q; }

    public boolean isReleasable() {
        return mItem != null || (mItem = mQueue.poll()) != null;
    }

    public boolean block() throws InterruptedException {
        if (mItem == null) mItem = mQueue.take(); return true;
    }

    public E getItem() {
        return mItem;
    }
}
```

Handles a blocking queue
Applying the ManagedBlocker Interface

- This example applies a ManagedBlocker on a BlockingQueue (from Java docs)

```java
class QueueTaker<E> implements ManagedBlocker {
    final BlockingQueue<E> mQueue;
    volatile E mItem = null;

    QueueTaker(BlockingQueue<E> q) { mQueue = q; }

    public boolean isReleasable() {
        return mItem != null || (mItem = mQueue.poll()) != null;
    }

    public boolean block() throws InterruptedException {
        if (mItem == null) mItem = mQueue.take(); return true;
    }

    public E getItem() {
        return mItem;
    }
}
```

The blocking queue
Applying the ManagedBlocker Interface

- This example applies a ManagedBlocker on a BlockingQueue (from Java docs)

class QueueTaker<E> implements ManagedBlocker {
    final BlockingQueue<E> mQueue;
    volatile E mItem = null;

    QueueTaker(BlockingQueue<E> q) { mQueue = q; }

    public boolean isReleasable() {
        return mItem != null || (mItem = mQueue.poll()) != null;
    }

    public boolean block() throws InterruptedException {
        if (mItem == null) mItem = mQueue.take(); return true;
    }

    public E getItem() {
        return mItem;
    }
}
Applying the ManagedBlocker Interface

- This example applies a ManagedBlocker on a BlockingQueue (from Java docs)

```java
class QueueTaker<E> implements ManagedBlocker {
    final BlockingQueue<E> mQueue;
    volatile E mItem = null;

    QueueTaker(BlockingQueue<E> q) { mQueue = q; }

    public boolean isReleasable() {
        return mItem != null || (mItem = mQueue.poll()) != null;
    }

    public boolean block() throws InterruptedException {
        if (mItem == null) mItem = mQueue.take(); return true;
    }

    public E getItem() {
        return mItem;
    }
}
```

*Block until an item is available*
Applying the ManagedBlocker Interface

- This example applies a ManagedBlocker on a BlockingQueue (from Java docs)

class QueueTaker<E> implements ManagedBlocker {
  final BlockingQueue<E> mQueue;
  volatile E mItem = null;

  QueueTaker(BlockingQueue<E> q) { mQueue = q; }

  public boolean isReleasable() {
    return mItem != null || (mItem = mQueue.poll()) != null;
  }

  public boolean block() throws InterruptedException {
    if (mItem == null) mItem = mQueue.take(); return true;
  }

  public E getItem() {
    return mItem;
  }
}
Encapsulating ManagedBlocker w/the BlockingTask Class
Encapsulating ManagedBlocker w/the BlockingTask Class

• BlockingTask integrates blocking Suppliers with the common fork/join pool

```java
class BlockingTask {
    ...
    public static<T> T callInManagedBlocker(Supplier<T> supplier) {
        SupplierManagedBlocker<T> managedBlocker =
            new SupplierManagedBlocker<T>(supplier);
        ...
        ForkJoinPool.managedBlock (managedBlocker);
        ...
        return managedBlocker.getResult();
    }
    ...
}
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex20
Encapsulating ManagedBlocker w/the BlockingTask Class

- BlockingTask integrates blocking Suppliers with the common fork/join pool

```java
public class BlockingTask {
    ...
    public static<T> T callInManagedBlocker(Supplier<T> supplier) {
        SupplierManagedBlocker<T> managedBlocker =
            new SupplierManagedBlocker<T>(supplier);
        ...
        ForkJoinPool.managedBlock(managedBlocker);
        ...
        return managedBlocker.getResult();
    }
    ...
}
```

Enables the use of blocking Suppliers with the common Java fork/join thread pool

See stackoverflow.com/q/37512662 for pros & cons of this approach
Encapsulating ManagedBlocker w/the BlockingTask Class

- BlockingTask integrates blocking Suppliers with the common fork/join pool

```java
public class BlockingTask {

    public static<T> T callInManagedBlocker(Supplier<T> supplier) {  

        SupplierManagedBlocker<T> managedBlocker =
            new SupplierManagedBlocker<T>(supplier);

        ForkJoinPool.managedBlock(managedBlocker);

        return managedBlocker.getResult();
    }

    ...

```

Create a helper object to encapsulate the supplier
Encapsulating ManagedBlocker w/the BlockingTask Class

- BlockingTask integrates blocking Suppliers with the common fork/join pool

```java
public class BlockingTask {
    ...
    public static<T> T callInManagedBlocker(Supplier<T> supplier){
        SupplierManagedBlocker<T> managedBlocker =
            new SupplierManagedBlocker<T>(supplier);
            ...
        ForkJoinPool.managedBlock(managedBlocker);
        ...
        return managedBlocker.getResult();
    }
    ...
}
```

Submit managedBlock to common ForkJoin thread pool
public class BlockingTask {
    ...
    public static<T> T callInManagedBlocker(Supplier<T> supplier) {
        SupplierManagedBlocker<T> managedBlocker =
            new SupplierManagedBlocker<T>(supplier);
        ...
        ForkJoinPool.managedBlock(managedBlocker);
        ...
        return managedBlocker.getResult();
    }
    ...
    Return the result of the blocking call
}

• BlockingTask integrates blocking Suppliers with the common fork/join pool

Encapsulating ManagedBlocker w/the BlockingTask Class
BlockingTask integrates blocking Suppliers with the common fork/join pool

```java
public class BlockingTask {
  ...
  private static class SupplierManagedBlocker<T>
      implements ForkJoinPool.ManagedBlocker {
    /** The blocking supplier. */
    private final Supplier<T> mSupplier;
    /** Keeps track of whether blocking supplier is done. */
    private boolean mDone = false;
    /** Result obtained from supplier. */
    private T mResult;

    /** Constructor initializes the field. */
    private SupplierManagedBlocker(final Supplier supplier) {
        mSupplier = supplier;
    }
  }

  Blocking Supplier work w/common fork/join pool
```

Encapsulating ManagedBlocker w/the BlockingTask Class
• BlockingTask integrates blocking Suppliers with the common fork/join pool

```java
public class BlockingTask {

    private static class SupplierManagedBlocker<T> implements ForkJoinPool.ManagedBlocker {
        /** The blocking supplier. */
        private final Supplier<T> mSupplier;
        /** Keeps track of whether blocking supplier is done. */
        private boolean mDone = false;
        /** Result obtained from supplier. */
        private T mResult;

        /** Constructor initializes the field. */
        private SupplierManagedBlocker(final Supplier supplier) {
            mSupplier = supplier;
        }
    }

    ...
BlockingTask integrates blocking Suppliers with the common fork/join pool

```java
public class BlockingTask {
    ...
    private static class SupplierManagedBlocker<T>
            implements ForkJoinPool.ManagedBlocker {
        ...
        public boolean block() {
            mResult = mSupplier.get(); mDone = true; return true; }
        
        public boolean isReleasable() {
            return mDone; }
        
        public T getResult() {
            return mResult; }
    }
}
```

Sets result via the blocking Supplier’s get() method
Encapsulating ManagedBlocker w/the BlockingTask Class

• BlockingTask integrates blocking Suppliers with the common fork/join pool

```java
public class BlockingTask {
    ...
    private static class SupplierManagedBlocker<T> implements ForkJoinPool.ManagedBlocker {
        ...
        public boolean block() {
            mResult = mSupplier.get(); mDone = true; return true; }
        
        public boolean isReleasable() {
            return mDone;
        }
        
        public T getResult() {
            return mResult;
        }
    }
}
```

True if blocking supplier has finished, else false
BlockingTask integrates blocking Suppliers with the common fork/join pool

```java
public class BlockingTask {
  ...
  private static class SupplierManagedBlocker<T>
      implements ForkJoinPool.ManagedBlocker {
    ...
    public boolean block()
    { mResult = mSupplier.get(); mDone = true; return true; } 

    public boolean isReleasable()
    { return mDone; }

    public T getResult()
    { return mResult; }
  }
```

- Returns the supplier’s result
Encapsulating ManagedBlocker w/the BlockingTask Class

- This example uses BlockingTask to ensure there are enough threads in the common thread pool.

```java
Image blockingDownload(URL url) {
    return BlockingTask
        .callInManagedBlocker
        (() -> downloadImage(url));
}
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex20](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex20)
Encapsulating ManagedBlocker w/the BlockingTask Class

- This example uses BlockingTask to ensure there are enough threads in the common thread pool

```java
Image blockingDownload(URL url) {
    return BlockingTask
        .callInManagedBlocker
        (() -> downloadImage(url));
}
```

*Transform a URL to an Image by downloading each image via its URL*
This example uses BlockingTask to ensure there are enough threads in the common thread pool. This call ensures the common fork/join thread pool is expanded to handle the blocking image download.
Encapsulating ManagedBlocker w/the BlockingTask Class

- This example uses BlockingTask to ensure there are enough threads in the common thread pool.
- Extra threads in the common fork-join pool are automatically terminated later.

```java
Image blockingDownload(URL url) {
    return BlockingTask
        .callInManagedBlocker
        (()-> downloadImage(url));
}
```
Encapsulating ManagedBlocker w/the BlockingTask Class

- This example uses BlockingTask to ensure there are enough threads in the common thread pool
- Extra threads in the common fork-join pool are automatically terminated later
- However, it’s possible to saturate the CPU cores during bursty workloads

```java
Image blockingDownload(URL url) {
    return BlockingTask.
callInManagedBlocker
    (()-> downloadImage(url));
}
```
End of the Java Fork-Join Pool Framework (Part 7)