Overview of the Java ThreadPoolExecutor

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

- Recognize the powerful features defined in the Java ExecutorService interface
- Understand other interfaces related to ExecutorService
- Know the key methods provided by ExecutorService
- Be aware of how ThreadPoolExecutor implements ExecutorService
- Focusing on how to customize ThreadPoolExecutor behaviors based on params passed to its constructor
Overview of the Java ThreadPoolExecutor
Overview of the Java ThreadPoolExecutor

- ThreadPoolExecutor implements the ExecutorService interface
- Indirectly via the AbstractExecutorService super class
Overview of the Java ThreadPoolExecutor

- ThreadPoolExecutor runs each submitted task via a worker thread provided by a pool.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html
Overview of the Java ThreadPoolExecutor

- ThreadPoolExecutor runs each submitted task via a worker thread provided by a pool.

A blocking queue passes tasks to threads.
Overview of the Java ThreadPoolExecutor

- ThreadPoolExecutor’s constructor can be configured via various parameters

```java
ThreadPoolExecutor
    (int corePoolSize,
     int maximumPoolSize,
     long keepAliveTime,
     TimeUnit unit,
     BlockingQueue<Runnable>
     workQueue,
     ThreadFactory
     threadFactory)
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#ThreadPoolExecutor
Overview of the Java ThreadPoolExecutor

- The # of threads in the pool can be controlled programmatically
- `corePoolSize` – # of threads to keep in the pool, even if they are idle
- `maximumPoolSize` – maximum # of threads to allow in the pool

```
ThreadPoolExecutor
(int corePoolSize,
 int maximumPoolSize,
 long keepAliveTime,
 TimeUnit unit,
 BlockingQueue<Runnable>
 workQueue,
 ThreadFactory
 threadFactory)
```
Overview of the Java ThreadPoolExecutor

- The lifetime of threads in the pool can be controlled programmatically
  - `keepAliveTime` – maximum time that excess idle threads will wait for new tasks before terminating when # of threads is greater than the core
  - `unit` – the time unit for the `keepAliveTime` argument

```
ThreadPoolExecutor
    (int corePoolSize,
     int maximumPoolSize,
     long keepAliveTime,
     TimeUnit unit,
     BlockingQueue<Runnable>
          workQueue,
     ThreadFactory
      threadFactory)
```
The queue holding tasks submitted by the `execute()` & `submit()` methods can be controlled programmatically.

- `workQueue` – the queue to use for holding tasks before they are run.

```java
ThreadPoolExecutor
(int corePoolSize,
 int maximumPoolSize,
 long keepAliveTime,
 TimeUnit unit,
 BlockingQueue<Runnable>
 workQueue,
 ThreadFactory
 threadFactory)
```
Overview of the Java ThreadPoolExecutor

- The queue can be strategized
- Direct handoff (used by cached pool)

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/SynchronousQueue.html
Overview of the Java ThreadPoolExecutor

- The queue can be strategized
- Direct handoff (used by cached pool)

See en.wikipedia.org/wiki/Concurrent_computing

```java
ThreadPoolExecutor
(int corePoolSize,
 int maximumPoolSize,
 long keepAliveTime,
 TimeUnit unit,
 BlockingQueue<Runnable>
 workQueue,
 ThreadFactory
 threadFactory)

SynchronousQueue implements a “rendezvous” model, where the caller blocks until the task is handed off to new or cached thread
```
Overview of the Java ThreadPoolExecutor

- The queue can be strategized
- Direct handoff (used by cached pool)
  - Pros – Avoids deadlocking on internal dependencies

See asznajder.github.io/thread-pool-induced-deadlocks
Overview of the Java ThreadPoolExecutor

- The queue can be strategized
- Direct handoff (used by cached pool)
  - Pros – Avoids deadlocking on internal dependencies
  - Cons – Can create unlimited threads

ThreadPoolExecutor
(int corePoolSize,
int maximumPoolSize,
long keepAliveTime,
TimeUnit unit,
BlockingQueue<Runnable> workQueue,
ThreadFactory threadFactory)
Overview of the Java ThreadPoolExecutor

- The queue can be strategized
- Direct handoff
- Unbounded queues (used by default fixed pool)

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/LinkedBlockingQueue.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/LinkedBlockingQueue.html)
Overview of the Java ThreadPoolExecutor

• The queue can be strategized
  • Direct handoff
• Unbounded queues (used by default fixed pool)
  • Pros – Smooths bursty requests

ThreadPoolExecutor
(int corePoolSize,
 int maximumPoolSize,
 long keepAliveTime,
 TimeUnit unit,
 BlockingQueue<Runnable> workQueue,
 ThreadFactory threadFactory)
Overview of the Java ThreadPoolExecutor

- The queue can be strategized
  - Direct handoff
- Unbounded queues (used by default fixed pool)
  - Pros – Smoothes bursty requests
  - Cons – Can consume unlimited resources

```java
ThreadPoolExecutor
(int corePoolSize,
 int maximumPoolSize,
 long keepAliveTime,
 TimeUnit unit,
 BlockingQueue<Runnable>
 workQueue,
 ThreadFactory
threadFactory)
```
Overview of the Java ThreadPoolExecutor

- The queue can be strategized
  - Direct handoff
  - Unbounded queues
  - Bounded queues (used by custom fixed pool)

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ArrayBlockingQueue.html
Overview of the Java ThreadPoolExecutor

- The queue can be strategized
  - Direct handoff
  - Unbounded queues
- Bounded queues (used by custom fixed pool)
  - Pros – Limits resource utilization

**ThreadPoolExecutor**

```
(int corePoolSize,
 int maximumPoolSize,
 long keepAliveTime,
 TimeUnit unit,
 BlockingQueue<Runnable> workQueue,
 ThreadFactory threadFactory)
```
Overview of the Java ThreadPoolExecutor

- The queue can be strategized
- Direct handoff
- Unbounded queues
- Bounded queues (used by custom fixed pool)
  - Pros – Limits resource utilization
  - Cons – Hard to tune & may deadlock

See [asznejder.github.io/thread-pool-induced-deadlocks](https://asznejder.github.io/thread-pool-induced-deadlocks)
Overview of the Java ThreadPoolExecutor

- The factory used to create threads can be controlled programmatically
- `threadFactory` – the factory to use when creating a new thread

```java
ThreadPoolExecutor
(int corePoolSize,
 int maximumPoolSize,
 long keepAliveTime,
 TimeUnit unit,
 BlockingQueue<Runnable>
 workQueue,
 ThreadFactory
 threadFactory)
```

ThreadFactory removes hardwiring of calls to new Thread, enabling programs to use special thread subclasses, priorities, etc.
End of Overview of the Java ThreadPoolExecutor