Java Concurrency Tutorial – Thread Pools

javacodegeeks.com Byron Kiourtzoglou September 16th, 2011 view original

One of the most generally useful concurrency enhancements delivered in Java 1.5 was the introduction of customizable thread pools. These thread pools give you quite a bit of control over things such as number of threads, reuse of threads, scheduling and thread construction. Let's review these.
First, thread pools. Let's dive right into **java.util.concurrent.ExecutorService**, which provides us the basic interface for a thread pool. All thread pools allow submitting **Callable** or **Runnable** instances for future execution. They also provide various pool management methods.

Pool Management

Various management methods exist for the pools. You can **shutdown()** the pool, which will reject any future submissions but complete processing of in-process executions and even those that had not yet started but were submitted before the shutdown was initiated. You can also more aggressively perform a shutdownNow(). This will also prevent any future submissions, but it has a few different, notable behaviours. It will not start execution of submitted but unstarted tasks. They will be in the returned list. It will also attempt to stop, or more precisely, **Thread.interrupt()** currently executing tasks. This is a best effort with no guarantee that these tasks will be successfully interrupted.

ThreadFactory

In a moment we will get into the **java.util.concurrent.Executors** builder class which can create various thread pool configurations, but first let's focus for a second on using ThreadFactory. You'll want to take advantage of **ThreadFactory** support in **Executors** and be in the habit of providing your own. The default **ThreadFactory** will give you give you an incrementing numbered pool naming scheme which is not all that helpful in logs or other monitoring. For the first pool created you'll get threads named *pool-1-thread-1*, *pool-1-thread-2* and the second one starts with *pool-2-thread-1*, etc. By providing your own

ThreadFactory, you can have threads named like *ReportProcessingThread1* and *HttpThread1*. Here's a simple example:

```
private AtomicLong counter = new AtomicLong();
private String name;
public Thread newThread(Runnable r) {
        Thread t = new Thread(r);
        t.setName(name + counter.incrementAndGet());
        return t;
}
```

ThreadFactory will only be called when a new **Thread** is created. Given that the JDK thread pools will reuse threads whenever possible, this class cannot be used to manage the beginning of execution.
Executors Builder Methods

Now back to the **Executors** utility builder methods. They are:

- **newCachedThreadPool()** will give you a thread pool that will reuse threads when possible, creating new ones as needed with no configured limit.
- **newFixedThreadPool(int nThreads)** will give you a thread pool that will use only up to the number of threads specified but will accept as many tasks as submitted for execution running them in submission order.
- **newScheduledThreadPool(int corePoolSize)** is used specifically for scheduling threads with delayed execution, on a recurring schedule on with recurring delay. The returned thread pool implements **ScheduledExecutorService** which exposes the additional scheduling methods **schedule(Runnable command, long delay, TimeUnit unit)**, **scheduleAtFixedRate(Runnable command, long initialDelay, long period, TimeUnit unit)** and **scheduleWithFixedDelay(Runnable command, long initialDelay, long delay, TimeUnit unit)**.
- **newSingleThreadExecutor()** and **newSingleThreadScheduledExecutor()**. These impose no limit on the number of tasks that can be submitted, only ensuring that a single thread/task is executing at a time.

Finally, there are a few helper methods for creating **Callable** instances from **Runnable**. This gets us into the newly created constructs for allowing threads to throw **Exceptions** and return values, something we had to work around quite painfully before. We'll consider these and how they are used with these thread pools in our next post.

Reference: Java Concurrency Part 3 – Thread Pools from our JCG partners at the Carfey Software blog.