

The Java™ Tutorials

Trail: Essential Java Classes

Lesson: Concurrency

Section: High Level Concurrency Objects

Subsection: Executors

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.

See [Java Language Changes](#) for a summary of updated language features in Java SE 9 and subsequent releases.

See [JDK Release Notes](#) for information about new features, enhancements, and removed or deprecated options for all JDK releases.

Thread Pools

Most of the executor implementations in `java.util.concurrent` use *thread pools*, which consist of *worker threads*. This kind of thread exists separately from the `Runnable` and `Callable` tasks it executes and is often used to execute multiple tasks.

Using worker threads minimizes the overhead due to thread creation. Thread objects use a significant amount of memory, and in a large-scale application, allocating and deallocating many thread objects creates a significant memory management overhead.

One common type of thread pool is the *fixed thread pool*. This type of pool always has a specified number of threads running; if a thread is somehow terminated while it is still in use, it is automatically replaced with a new thread. Tasks are submitted to the pool via an internal queue, which holds extra tasks whenever there are more active tasks than threads.

An important advantage of the fixed thread pool is that applications using it *degrade gracefully*. To understand this, consider a web server application where each HTTP request is handled by a separate thread. If the application simply creates a new thread for every new HTTP request, and the system receives more requests than it can handle immediately, the application will suddenly stop responding to *all* requests when the overhead of all those threads exceed the capacity of the system. With a limit on the number of the threads that can be created, the application will not be servicing HTTP requests as quickly as they come in, but it will be servicing them as quickly as the system can sustain.

A simple way to create an executor that uses a fixed thread pool is to invoke the `newFixedThreadPool` factory method in `java.util.concurrent.Executors`. This class also provides the following factory methods:

- The `newCachedThreadPool` method creates an executor with an expandable thread pool. This executor is suitable for applications that launch many short-lived tasks.
- The `newSingleThreadExecutor` method creates an executor that executes a single task at a time.
- Several factory methods are `ScheduledExecutorService` versions of the above executors.

If none of the executors provided by the above factory methods meet your needs, constructing instances of `java.util.concurrent.ThreadPoolExecutor` or `java.util.concurrent.ScheduledThreadPoolExecutor` will give you additional options.

[About Oracle](#) | [Contact Us](#) | [Legal Notices](#) | [Terms of Use](#) | [Your Privacy Rights](#)

Copyright © 1995, 2022 Oracle and/or its affiliates. All rights reserved.

Previous page: Executor Interfaces

Next page: Fork/Join