Day 48

Daemon Thread in Java

**Daemon thread in Java** is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc.

You can see all the detail by typing the jconsole in the command prompt. The jconsole tool provides information about the loaded classes, memory usage, running threads etc.

Points to remember for Daemon Thread in Java

* It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
* Its life depends on user threads.
* It is a low priority thread.

Why JVM terminates the daemon thread if there is no user thread?

The sole purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, why should JVM keep running this thread. That is why JVM terminates the daemon thread if there is no user thread.

Methods for Java Daemon thread by Thread class

The java.lang.Thread class provides two methods for java daemon thread.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1) | public void setDaemon(boolean status) | is used to mark the current thread as daemon thread or user thread. |
| 2) | public boolean isDaemon() | is used to check that current is daemon. |

# Java Thread Pool

**Java Thread pool** represents a group of worker threads that are waiting for the job and reused many times.

In the case of a thread pool, a group of fixed-size threads is created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, the thread is contained in the thread pool again.

## Thread Pool Methods

**newFixedThreadPool(int s):** The method creates a thread pool of the fixed size s.

**newCachedThreadPool():** The method creates a new thread pool that creates the new threads when needed but will still use the previously created thread whenever they are available to use.

**newSingleThreadExecutor():** The method creates a new thread.

### Advantage of Java Thread Pool

**Better performance** It saves time because there is no need to create a new thread.

### Real time usage

It is used in Servlet and JSP where the container creates a thread pool to process the request.

**Explanation:** It is evident by looking at the output of the program that tasks 4 and 5 are executed only when the thread has an idle thread. Until then, the extra tasks are put in the queue.

The takeaway from the above example is when one wants to execute 50 tasks but is not willing to create 50 threads. In such a case, one can create a pool of 10 threads. Thus, 10 out of 50 tasks are assigned, and the rest are put in the queue. Whenever any thread out of 10 threads becomes idle, it picks up the 11thtask. The other pending tasks are treated the same way.

## Risks involved in Thread Pools

The following are the risk involved in the thread pools.

**Deadlock:** It is a known fact that deadlock can come in any program that involves multithreading, and a thread pool introduces another scenario of deadlock. Consider a scenario where all the threads that are executing are waiting for the results from the threads that are blocked and waiting in the queue because of the non-availability of threads for the execution.

**Thread Leakage:** Leakage of threads occurs when a thread is being removed from the pool to execute a task but is not returning to it after the completion of the task. For example, when a thread throws the exception and the pool class is not able to catch this exception, then the thread exits and reduces the thread pool size by 1. If the same thing repeats a number of times, then there are fair chances that the pool will become empty, and hence, there are no threads available in the pool for executing other requests.

**Resource Thrashing:** A lot of time is wasted in context switching among threads when the size of the thread pool is very large. Whenever there are more threads than the optimal number may cause the starvation problem, and it leads to resource thrashing.

## Points to Remember

Do not queue the tasks that are concurrently waiting for the results obtained from the other tasks. It may lead to a deadlock situation, as explained above.

Care must be taken whenever threads are used for the operation that is long-lived. It may result in the waiting of thread forever and will finally lead to the leakage of the resource.

In the end, the thread pool has to be ended explicitly. If it does not happen, then the program continues to execute, and it never ends. Invoke the shutdown() method on the thread pool to terminate the executor. Note that if someone tries to send another task to the executor after shutdown, it will throw a RejectedExecutionException.

One needs to understand the tasks to effectively tune the thread pool. If the given tasks are contrasting, then one should look for pools for executing different varieties of tasks so that one can properly tune them.

To reduce the probability of running JVM out of memory, one can control the maximum threads that can run in JVM. The thread pool cannot create new threads after it has reached the maximum limit.

A thread pool can use the same used thread if the thread has finished its execution. Thus, the time and resources used for the creation of a new thread are saved.

## Tuning the Thread Pool

The accurate size of a thread pool is decided by the number of available processors and the type of tasks the threads have to execute. If a system has the P processors that have only got the computation type processes, then the maximum size of the thread pool of P or P + 1 achieves the maximum efficiency. However, the tasks may have to wait for I/O, and in such a scenario, one has to take into consideration the ratio of the waiting time (W) and the service time (S) for the request; resulting in the maximum size of the pool P \* (1 + W / S) for the maximum efficiency.

## Conclusion

A thread pool is a very handy tool for organizing applications, especially on the server-side. Concept-wise, a thread pool is very easy to comprehend. However, one may have to look at a lot of issues when dealing with a thread pool. It is because the thread pool comes with some risks involved it (risks are discussed above).