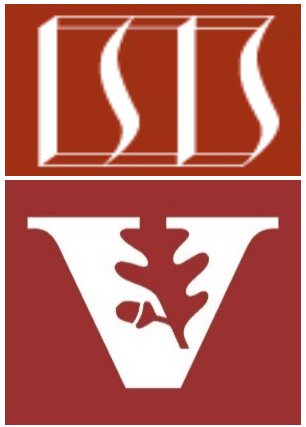
Programming with Java

TaskPerThreadExecutor

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 Java's structured concurrency model
- Recognize the classes used to program Java's structure concurrency model, e.g.
  - ThreadPerTaskExecutor

```
try (var executor = Executors
      .newVirtualThreadPerTaskExecutor()) {
    IntStream
      .range(0, 1_000_000)

      .forEach(i -> executor
         .submit(() -> {
            Thread.sleep(Duration
                        .ofSeconds(1));
            return i;
          }));
}
```

Programming with Java ThreadPerTaskExecutor

Programming with Java ThreadPerTaskExecutor

- This feature adds two new factory methods in the Java Executors utility class & extends the ExecutorService interface

```
static ExecutorService    newThreadPerTaskExecutor
                          (ThreadFactory threadFactory)
```

Preview.

Creates an Executor that starts a new Thread for each task.

```
static ExecutorService    newVirtualThreadPerTaskExecutor()
```

Preview.

Creates an Executor that starts a new virtual Thread for each task.

```
public interface ExecutorService
extends Executor, AutoCloseable
```

An `Executor` that provides methods to manage termination and methods that can produce a `Future` for tracking progress of one or more asynchronous tasks.

See docs.oracle.com/en/java/javase/19/docs/api/java.base/java/util/concurrent/Executors.html

Programming with Java ThreadPerTaskExecutor

- This feature adds two new factory methods in the Java Executors utility class & extends the ExecutorService interface
  - The newThreadPerTaskExecutor() factory method starts a new Thread for each task
    - The type of the Thread can be designated via a ThreadFactory

newThreadPerTaskExecutor

```
public static ExecutorService newThreadPerTaskExecutor
(ThreadFactory threadFactory)
```

newThreadPerTaskExecutor is a preview API of the Java platform.
Programs can only use newThreadPerTaskExecutor when preview features are enabled.
Preview features may be removed in a future release, or upgraded to permanent features of the Java platform.

Creates an Executor that starts a new Thread for each task. The number of threads created by the Executor is unbounded.

Invoking `cancel(true)` on a `Future` representing the pending result of a task submitted to the Executor will `interrupt` the thread executing the task.

Parameters:

`threadFactory` - the factory to use when creating new threads

Returns:

a new executor that creates a new Thread for each task

See java/util/concurrent/Executors.html#newThreadPerTaskExecutor

Programming with Java ThreadPerTaskExecutor

- This feature adds two new factory methods in the Java Executors utility class & extends the ExecutorService interface
  - The newThreadPerTaskExecutor() factory method starts a new Thread for each task
  - The newVirtualThreadPerTask Executor() starts a new Java virtual Thread for each task

newVirtualThreadPerTaskExecutor

```
public
static ExecutorService newVirtualThreadPerTaskExecutor()
```

`newVirtualThreadPerTaskExecutor` is a preview API of the Java platform.
Programs can only use `newVirtualThreadPerTaskExecutor` when preview features are enabled.
Preview features may be removed in a future release, or upgraded to permanent features of the Java platform.

Creates an Executor that starts a new virtual Thread for each task. The number of threads created by the Executor is unbounded.

This method is equivalent to invoking `newThreadPerTaskExecutor(ThreadFactory)` PREVIEW with a thread factory that creates virtual threads.

Returns:

a new executor that creates a new virtual Thread for each task

Throws:

`UnsupportedOperationException` - if preview features are not enabled

See java/util/concurrent/Executors.html#newVirtualThreadPerTaskExecutor()

Programming with Java ThreadPerTaskExecutor

- These Executors are used with the Java try-with-resources feature

```
try (var executor = Executors
    .newVirtualThreadPerTaskExecutor()){
  IntStream
    .range(0, 10_000_000)
    .forEach(i -> executor
      .submit(() -> {
        Thread.sleep(Duration
                     .ofSeconds(1));
        return i;
      }));
}
```

Creates an Executor that starts a new virtual Thread for each task

See howtodoinjava.com/java/multi-threading/virtual-threads/#43-using-executorsnewvirtualthreadpertaskexecutor

Programming with Java ThreadPerTaskExecutor

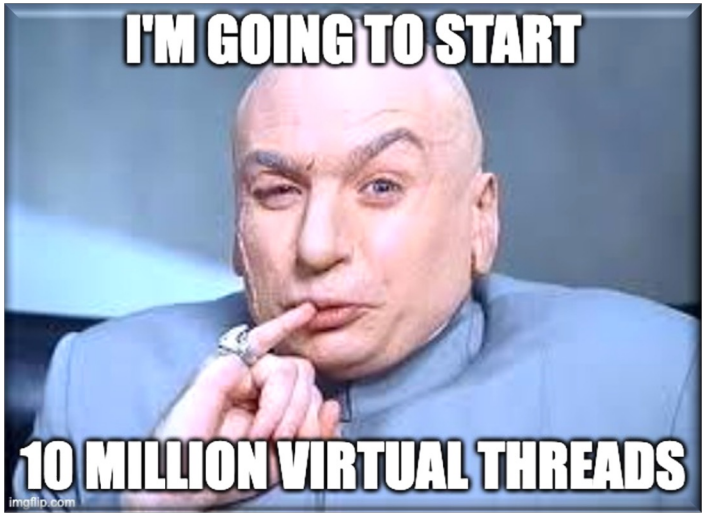
- These Executors are used with the Java try-with-resources feature

```
try (var executor = Executors
      .newVirtualThreadPerTaskExecutor()){
   IntStream
      .range(0, 10_000_000)
      .forEach(i -> executor
         .submit(() -> {
            Thread.sleep(Duration
                        .ofSeconds(1));
            return i;
         }));
}
```

Generate 10 million iterations

Programming with Java ThreadPerTaskExecutor

- These Executors are used with the Java try-with-resources feature




```
try (var executor = Executors
      .newVirtualThreadPerTaskExecutor()){
   IntStream
      .range(0, 10_000_000)
      .forEach(i -> executor
         .submit(() -> {
            Thread.sleep(Duration
                         .ofSeconds(1));
            return i;
         }));
}
```

Submit 10 million tasks, each of which is executed via a Java virtual Thread

Programming with Java ThreadPerTaskExecutor

- These Executors are used with the Java try-with-resources feature

```
try (var executor = Executors
      .newVirtualThreadPerTaskExecutor()){
   IntStream
      .range(0, 10_000_000)
      .forEach(i -> executor
         .submit(() -> {
            Thread.sleep(Duration
                        .ofSeconds(1));
            return i;
         }));
}
```

All these submitted virtual threads must complete by the end of the enclosing scope

Programming with Java ThreadPerTaskExecutor

- These Executors are used with the Java try-with-resources feature
  - This mechanism is simple, but also limited

```
try (var executor = Executors
      .newVirtualThreadPerTaskExecutor()){
   IntStream
      .range(0, 10_000_000)
      .forEach(i -> executor
         .submit(() -> {
            Thread.sleep(Duration
                        .ofSeconds(1));
            return i;
         }));
}
```

Programming with Java ThreadPerTaskExecutor

- These Executors are used with the Java try-with-resources feature
  - This mechanism is simple, but also limited
    - It lacks support for fine-grained exception handling, "invoke any" semantics, & automatic task cancellation

```
try (var executor = Executors
       .newVirtualThreadPerTaskExecutor()){
     IntStream
       .range(0, 10_000_000)
       .forEach(i -> executor
          .submit(() -> {
             Thread.sleep(Duration
                          .ofSeconds(1));
             return i;
          }));
```

Programming with Java ThreadPerTaskExecutor

- These Executors are used with the Java try-with-resources feature
  - This mechanism is simple, but also limited
    - It lacks support for fine-grained exception handling, "invoke any" semantics, & automatic task cancellation
  - However, it can serve as a "drop-in" replacement for common ExecutorService use-cases

```
@Bean(APPLICATION_TASK_
      EXECUTOR_BEAN_NAME) public
AsyncTaskExecutor asyncTaskExecutor(){
  return new TaskExecutorAdapter
    (Executors
     .newVirtualThreadPerTaskExecutor());
}
```

This Bean configures the Spring WebMVC platform so it will create a Java virtual thread to process each client request

See spring.io/blog/2022/10/11/embracing-virtual-threads

Programming with Java ThreadPerTaskExecutor

- These Executors are used with the Java try-with-resources feature
  - This mechanism is simple, but also limited
  - These limitations motivate the need for the new Java StructuredTaskScope

```
try (var scope = new
      StructuredTaskScope
      .ShutdownOnFailure()) {
  Supplier<String> user = scope
    .fork(() -> findUser());
  Supplier<Integer> order = scope
    .fork(() -> fetchOrder());

  scope.join();
  scope.throwIfFailed();

  return new Response
                (user.get(),
                 order.get());
}
```

See upcoming lesson on "*Programming with Java StructuredTaskScope*"

End of Programming with Java TaskPerThreadExecutor