Programming with Java

StructuredTaskScope

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
  - StructuredTaskScope
  - Both ShutdownOnFailure & ShutdownOnSuccess

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

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

    return new Response
        (user.resultNow(),
         order.resultNow());
}
```
Programming with Java StructuredTaskScope
StructuredTaskScope is the basic API for Java structured concurrency

**Class StructuredTaskScope<T>**

java.lang.Object  
jdk.incubator.concurrent.StructuredTaskScope<T>

**Type Parameters:**
T - the result type of tasks executed in the scope

**All Implemented Interfaces:**
AutoCloseable

**Direct Known Subclasses:**
StructuredTaskScope.ShutdownOnFailure,  
StructuredTaskScope.ShutdownOnSuccess

```java
public class StructuredTaskScope<T>  
extends Object  
implements AutoCloseable
```

A basic API for structured concurrency. StructuredTaskScope supports cases where a task splits into several concurrent subtasks, to be executed in their own threads, and where the subtasks must complete before the main task continues. A StructuredTaskScope can be used to ensure that the lifetime of a concurrent operation is confined by a syntax block, just like that of a sequential operation in structured programming.

See [jdk/incubator/concurrent/StructuredTaskScope.html](jdk/incubator/concurrent/StructuredTaskScope.html)
StructuredTaskScope is the basic API for Java structured concurrency

- It splits a task into several subtasks that run concurrently within a syntax block

See jdk/incubator/concurrent/StructuredTaskScope.html
StructuredTaskScope is the basic API for Java structured concurrency

- It splits a task into several subtasks that run concurrently within a syntax block
- It defines several nested subclasses
StructuredTaskScope is the basic API for Java structured concurrency

- It splits a task into several subtasks that run concurrently within a syntax block
- It defines several nested subclasses
  - ShutdownOnFailure
    - Captures the exception of the first subtask to complete abnormally

StructuredTaskScope is the basic API for Java structured concurrency

- It splits a task into several subtasks that run concurrently within a syntax block
- It defines several nested subclasses
  - ShutdownOnFailure
  - ShutdownOnSuccess
- Captures the result of the first subtask to complete successfully

StructuredTaskScope is the basic API for Java structured concurrency

• It splits a task into several subtasks that run concurrently within a syntax block

• It defines several nested subclasses
  • ShutdownOnFailure
  • ShutdownOnSuccess
    • Captures the result of the first subtask to complete successfully
      • Essentially like “invokeAny()”
Programming with Java
ShutdownOnFailure
• ShutdownOnFailure is used with the try-with-resources feature, like the Executors.ThreadPerTaskExecutor

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

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

    return new Response
        (user.resultNow(),
         order.resultNow());
}
```
Programming with Java ShutdownOnFailure

- ShutdownOnFailure is used with the try-with-resources feature, like the Executors.ThreadPerTaskExecutor

```java
try (var scope = new StructuredTaskScope
     .ShutdownOnFailure()) {
    Future<String> user = scope
        .fork(() -> findUser());
    Future<Integer> order = scope
        .fork(() -> fetchOrder());
    scope.join();
    scope.throwIfFailed();
    return new Response
        (user.resultNow(),
         order.resultNow());
}
```

See [jdk/incubator/concurrent/StructuredTaskScope.html#fork](https://jdk/incubator/concurrent/StructuredTaskScope.html#fork)

Creates a new virtual Thread every time fork() is called
ShutdownOnFailure is used with the try-with-resources feature, like the Executors.ThreadPerTaskExecutor. However, it’s more flexible due to its join() method.

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

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

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

This barrier synchronizer waits for all threads to finish or for the task scope to shut down if an exception should occur.
Programming with Java ShutdownOnFailure

- ShutdownOnFailure is used with the try-with-resources feature, like the Executors.ThreadPerTaskExecutor
- However, it’s more flexible due to its join() method
- It can also handle any exceptions that arise

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

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

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

Throws an Exception if a sub-task completed abnormally

See [jdk/incubator/concurrent/StructuredTaskScope.ShutdownOnFailure.html#throwIfFailed](https://jdk/incubator/concurrent/StructuredTaskScope.ShutdownOnFailure.html#throwIfFailed)
Programming with Java ShutdownOnFailure

- **ShutdownOnFailure** is used with the try-with-resources feature, like the Executors.ThreadPerTaskExecutor.
- However, it’s more flexible due to its join() method.
- It can also handle any exceptions that arise.
- Users can access Future results without blocking.

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

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

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

See [java/util/concurrent/Future.html#resultNow](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html#resultNow)

*Return a result using new Future methods*
Programming with Java

ShutdownOnSuccess
• ShutdownOnSuccess also uses try-with-resources

```java
try (var scope = new StructuredTaskScope.ShutdownOnSuccess
    <List<BigFraction>>()
) {
    var quickSortF = scope
        .fork(() -> quickSort(list));

    var heapSortF = scope
        .fork(() -> heapSort(list));

    scope.join();

    return scope.result();
}
```

See [howtodoinjava.com/java/multi-threading/structured-concurrency](http://howtodoinjava.com/java/multi-threading/structured-concurrency)
Programming with Java ShutdownOnSuccess

- ShutdownOnSuccess also uses try-with-resources
- It provides “invoke-any” semantics that take only the fastest result

```java
try (var scope = new StructuredTaskScope.ShutdownOnSuccess
     .<List<BigFraction>>()
     .fork(() -> quickSort(list)))

    var quickSortF = scope
        .fork(() -> quickSort(list));

    var heapSortF = scope
        .fork(() -> heapSort(list));

    scope.join();

    return scope.result();
}
```

See howtodoinjava.com/java/multi-threading/executorservice-invokeany
Programming with Java ShutdownOnSuccess

- ShutdownOnSuccess also uses try-with-resources
- It provides “invoke-any” semantics that take only the fastest result

```java
try (var scope = new StructuredTaskScope.<List<BigFraction>>().ShutdownOnSuccess()) {
    var quickSortF = scope.fork(() -> quickSort(list));
    var heapSortF = scope.fork(() -> heapSort(list));

    scope.join();

    return scope.result();
}
```

Run quicksort & heapsort in parallel!

See [jdk/incubator/concurrent/StructuredTaskScope.html#fork](https://jdk/incubator/concurrent/StructuredTaskScope.html#fork)
Programming with Java ShutdownOnSuccess

- ShutdownOnSuccess also uses try-with-resources
- It provides “invoke-any” semantics that take only the fastest result

```java
try (var scope = new StructuredTaskScope.ShutdownOnSuccess
        .<List<BigFraction>>()
        {
    var quickSortF = scope
        .fork(() -> quickSort(list));

    var heapSortF = scope
        .fork(() -> heapSort(list));

    scope.join();

    return scope.result();
```

**Programming with Java ShutdownOnSuccess**

- ShutdownOnSuccess also uses try-with-resources
- It provides “invoke-any” semantics that take only the fastest result

```java
try (var scope = new StructuredTaskScope
     .ShutdownOnSuccess
     <List<BigFraction>>()
     ) {

    var quickSortF = scope
     .fork(() -> quickSort(list));

    var heapSortF = scope
     .fork(() -> heapSort(list));

    scope.join();

    return scope.result();
}
```

See [jdk/incubator/concurrent/StructuredTaskScope.ShutdownOnSuccess.html#result](jdk/incubator/concurrent/StructuredTaskScope.ShutdownOnSuccess.html#result)
End of Programming with Java StructuredTaskScope