3. Multi-Threading – Create a simple framework where work items can be submitted

Posted on March 9, 2016 by Arulkumaran Kumaraswamipillai

Q: Create a simple framework where work items can be submitted using Java 8 or later. Here are the use cases:

#1: A **work item** is an instance of a class.

#2: The definition of "**parallelism**" controls how many threads are created to execute the work items in parallel (aka **asynchronously**).

#3: The framework needs to ensure that the number of threads executing the work item should remain the same until

the threads finished executing all the work items.

#4: If work item dies, the framework should retry it.

#5: No need to cater for timeouts.

Use the following interfaces

E.g. Take 5 work items and execute 3 in parallel (i.e. 3 asynchronously).

```
1
2  package com.homeassignment.task2;
3
4  import java.util.List;
5  import java.util.concurrent.ExecutionException;
6
7  public interface WorkItemExecutor
8
9  {
10      void executeWorkItem(List<WorkItem> w, int p
11 }
12
13
```

E.g. Work item execution. Takes a "WorkItemCompletionCallback", which is called on completion of a work item.

```
1
2  package com.homeassignment.task2;
3
4  public interface WorkItem
5
6  {
7      void execute(WorkItemCompletionCallback callb
8  }
9
```

Callback logic.

```
1
2  package com.homeassignment.task2;
3
4  public interface WorkItemCompletionCallback
5
6  {
7      void complete(String name);
8  }
```

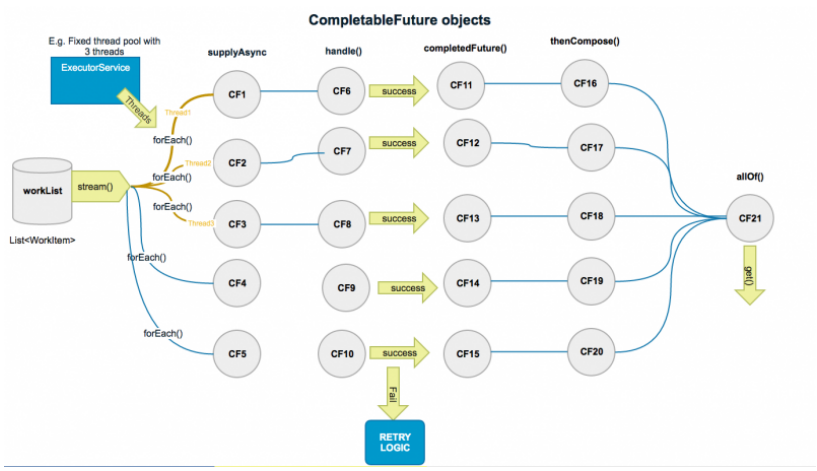

16 Technical Key Areas

```
9
```

A: Start implementing the logic to create a framework to execute work items in parallel using Java 8 “ExecutorService” and “CompletableFuture” APIs. The basics are covered in ExecutorService Vs Fork/Join & Future Vs CompletableFuture Interview Q&A.

Step 1: WorkItemExecutorImpl that implements WorkItemExecutor

This implementation has the most of the logic, lambda expressions, use and chaining of CompletableFuture, retry logic, and use of ExecutorService thread pool.



Functional Reactive Programming depicting the CompletableFuture objects.

```
1
2  package com.homeassignment.task2;
3
4  import java.util.ArrayList;
5  import java.util.List;
6  import java.util.concurrent.CompletableFuture;
7  import java.util.concurrent.ExecutionException;
8  import java.util.concurrent.ExecutorService;
9  import java.util.concurrent.Executors;
10 import java.util.concurrent.TimeUnit;
11
12 import org.slf4j.Logger;
13 import org.slf4j.LoggerFactory;
```


⊞ Java Debugging (21)
⊞ Judging Experience In
⊞ Low Latency (7)
⊞ Memory Managemen
⊞ Performance (13)
⊞ QoS (8)
⊞ Scalability (4)
⊞ SDLC (6)
⊞ Security (13)
⊞ Transaction Managen

80+ step by step Java Tutorials

open all | close all
⊞ Setting up Tutorial (6)
⊞ Tutorial - Diagnosis (2
⊞ Akka Tutorial (9)
⊞ Core Java Tutorials (2
⊞ Hadoop & Spark Tuto
⊞ JEE Tutorials (19)
⊞ Scala Tutorials (1)
⊞ Spring & HIbernate Tu
⊞ Tools Tutorials (19)
⊞ Other Tutorials (45)

100+ Java pre-interview coding tests

open all | close all
⊞ Can you write code? (
⊞ ♦ Complete the given
⊞ Converting from A to I
⊞ Designing your classe
⊞ Java Data Structures
⊞ Passing the unit tests
⊞ What is wrong with th

```
14
15 public class WorkItemExecutorImpl implements Wor
16
17     private Logger LOG = LoggerFactory.getLogger
18
19     private ExecutorService concurrentThreadExec
20
21     @Override
22     public void executeWorkItem(List<WorkItem> w
23
24         //Pool of threads
25         concurrentThreadExecutor = Executors.new
26
27         //callback
28         WorkItemCompletionCallback callback = ne
29
30         List<CompletableFuture<Object>> listCf =
31
32         //run asynchronously on concurrentThread
33         workList.stream().forEach(wi -> {
34       CompletableFuture<Object> thenCompose = Co
35                 wi.execute(callback);
36                 return null;
37             }, concurrentThreadExecutor).handle(
38
39                 if (e == null) { // no error
40                         return CompletableFuture
41                     } else { // retry
42                         retry(wi, callback);
43                         return null;
44                     }
45
46                 }).thenCompose(x -> x);
47
48             listCf.add(thenCompose);
49
50         });
51
52         @SuppressWarnings("unchecked")
53         CompletableFuture<WorkItem>[] cfArray =
54
55         // executes after all work items have co
56         // of workList
57         CompletableFuture<Void> thenRun = Comple
58             callback.complete(workList.size() +
59         });
60
61         try {
62             thenRun.get(); // blocking call that
63         } catch (InterruptedException | Executio
64             LOG.error("" + e);
65         }
66
67         concurrentThreadExecutor.shutdown();
68
69     }
70
71     //handle failed executions by retrying x num
72     private void retry(WorkItem wi, WorkItemComp
73         int retries = 5;
74
75         for (int i = 1; i <= retries; i++) {
76             try {
77                 TimeUnit.SECONDS.sleep(1);
78                 LOG.error("Error processing " +
79                 wi.execute(callback);
```

Writing Code Home A
Written Test Core Jav
Written Test JEE (1)

How good are your .....?

open all | close all
Career Making Know-
Job Hunting & Resum

```
80                  return;
81              } catch (Throwable t) {
82                  LOG.error("" + t);
83              }
84          }
85          throw new RuntimeException("All " + retr
86      }
87
88  }
89
90
91
```

1) The **supplyAsync(….)** executes asynchronously and returns a new “CompletableFuture<Object>” stage object.

2) The **handle(..,..)** executes after supplyAsync(..) stage is completed and returns a new CompletableFuture<CompletableFuture<Object>> stage object. The given function is invoked with the result (or null if none) and the exception (or null if none) of this stage.

3) The **completedFuture(…)** returns a new “CompletableFuture<Object>” that is already returned with a given value.

4) The **thenCompose(…)** returns a new “CompletableFuture<Object>” when this stage completes normally. The main reason to do is because “**2)**” handle(…) returns of type “CompletableFuture<CompletableFuture<Object>>” and “thenCompose(..)” returns “CompletableFuture<Object>”.

5) The **allOf(…)** returns a new “CompletableFuture<Void>” that is completed when all of the given CompletableFutures (i.e. **listCf**) are complete.

6) The “**thenRun.get();**” is an optional blocking call that returns a result or throws an exception from the allOf(…) stage.

Step 2: WorkItemImpl that implements WorkItem

```
1
2  package com.homeassignment.task2;
3
4  import java.util.concurrent.TimeUnit;
5
6  import org.slf4j.Logger;
7  import org.slf4j.LoggerFactory;
8
9  public class WorkItemImpl implements WorkItem {
10
11      private Logger LOG = LoggerFactory.getLogger
12
13      private String name;
14
15      public WorkItemImpl(String name) {
16          this.name = name;
17      }
18
19      @Override
20      public void execute(WorkItemCompletionCallba
21
22          LOG.info("Started processing " + name);
23
24          try {
25              TimeUnit.SECONDS.sleep(3);
26          } catch (InterruptedException e) {
27              LOG.error("" + e);
28          }
29
30          callback.complete(name);
31
32      }
33
34      @Override
35      public String toString() {
36          return "WorkItemImpl [name=" + name + "]
37      }
38 }
39
40
```

Step 3: WorkItemCompletionCallbackImpl that implements WorkItemCompletionCallback

```
1
2  package com.homeassignment.task2;
3
4  import org.slf4j.Logger;
5  import org.slf4j.LoggerFactory;
6
7  public class WorkItemCompletionCallbackImpl impl
8
9      private Logger LOG = LoggerFactory.getLogger
10
11     @Override
```

```
12      public void complete(String name) {
13          LOG.info("completed processing " + name)
14      }
15  }
16
17
```

Step 4: Finally, a client to run standalone

Let's create 5 work items and run 3 of them inn parallel.

```
1
2   import java.util.ArrayList;
3   import java.util.List;
4   import java.util.concurrent.ExecutionException;
5
6   import com.homeassignment.task2.WorkItem;
7   import com.homeassignment.task2.WorkItemExecutor
8   import com.homeassignment.task2.WorkItemExecutor
9   import com.homeassignment.task2.WorkItemImpl;
10
11  public class WorkItemExecutorClient {
12
13      public static void main(String[] args) throw
14          WorkItemExecutor wie = new WorkItemExecu
15
16          List<WorkItem> workList = new ArrayList<
17
18          for (int i = 1; i <= 5; i++) {
19              WorkItem wi = new WorkItemImpl("Work
20              workList.add(wi);
21          }
22
23          wie.executeWorkItem(workList, 3);
24      }
25  }
26
27
```

Output:

```
1
2   14:15:17.549 [pool-2-thread-3] INFO  c.homeassig
3   14:15:17.549 [pool-2-thread-1] INFO  c.homeassig
4   14:15:17.549 [pool-2-thread-2] INFO  c.homeassig
5   14:15:20.557 [pool-2-thread-3] INFO  c.h.t.WorkI
6   14:15:20.557 [pool-2-thread-2] INFO  c.h.t.WorkI
7   14:15:20.557 [pool-2-thread-1] INFO  c.h.t.WorkI
8   14:15:20.558 [pool-2-thread-3] INFO  c.homeassig
9   14:15:20.558 [pool-2-thread-1] INFO  c.homeassig
10  14:15:23.562 [pool-2-thread-3] INFO  c.h.t.WorkI
11  14:15:23.562 [pool-2-thread-1] INFO  c.h.t.WorkI
12  14:15:23.562 [pool-2-thread-1] INFO  c.h.t.WorkI
13
```

Step 5: Unhappy path:

Let's purposely modify "WorkItemImpl" to throw exception to test the failure scenarios. A counter variable "count" was introduced and every multiples of 3, throw an error on purpose.

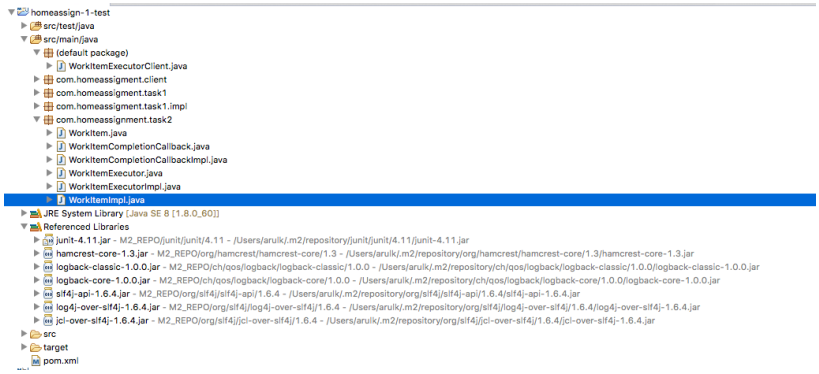
```
1
2  package com.homeassignment.task2;
3
4  import java.util.concurrent.TimeUnit;
5
6  import org.slf4j.Logger;
7  import org.slf4j.LoggerFactory;
8
9  public class WorkItemImpl implements WorkItem {
10
11     private Logger LOG = LoggerFactory.getLogger
12
13     private String name;
14
15     static int count = 0;
16
17     public WorkItemImpl(String name) {
18         this.name = name;
19     }
20
21     @Override
22     public void execute(WorkItemCompletionCallba
23
24         LOG.info("Started processing " + name);
25         ++count;
26
27         if (count % 3 == 0) {
28             throw new RuntimeException("Error pr
29         }
30
31         try {
32             TimeUnit.SECONDS.sleep(5);
33         } catch (InterruptedException e) {
34             LOG.error("" + e);
35         }
36
37         callback.complete(name);
38
39     }
40
41     @Override
42     public String toString() {
43         return "WorkItemImpl [name=" + name + "]
44     }
45 }
46
47
```

Step 6: Rerun for unhappy path:

Rerunning the “WorkItemExecutorClient” after the above change will yield an output as shown below.

```
1
2  14:20:08.525 [pool-2-thread-3] INFO  c.homeassig
3  14:20:08.525 [pool-2-thread-1] INFO  c.homeassig
4  14:20:08.525 [pool-2-thread-2] INFO  c.homeassig
5  14:20:09.534 [pool-2-thread-2] ERROR c.h.task2.W
6  14:20:09.534 [pool-2-thread-2] INFO  c.homeassig
7  14:20:13.531 [pool-2-thread-1] INFO  c.h.t.WorkI
8  14:20:13.531 [pool-2-thread-3] INFO  c.h.t.WorkI
9  14:20:13.532 [pool-2-thread-1] INFO  c.homeassig
10 14:20:13.532 [pool-2-thread-3] INFO  c.homeassig
11 14:20:14.534 [pool-2-thread-3] ERROR c.h.task2.W
12 14:20:14.535 [pool-2-thread-3] INFO  c.homeassig
13 14:20:14.537 [pool-2-thread-2] INFO  c.h.t.WorkI
14
```

The project structure:



Maven Project Structure

Popular Posts

♦ 11 Spring boot interview questions & answers

825 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

766 views

18 Java scenarios based interview Questions and Answers

400 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

388 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

295 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers

to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ 10: ♦ ExecutorService Vs Fork/Join & Future Vs CompletableFuture Interview Q&A

4a. 10-digit phone number to produce a list of words matching first letters of the phone number ›

Posted in member-paid, Reactive Programming, Writing Code Home Assignements

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer

 ↑