02: ♥♦ 10+ Java multi-threading scenarios interview Q&As

Posted on October 14, 2014 by Arulkumaran Kumaraswamipillai — 2 Comments ↓

Java multi-threading interview questions are very popular from beginner to experienced level candidates. Many struggle, including the so called experienced. That is why interviewers love this key area of concurrency management to assess your capabilities.

In Java, there are low level interfaces/classes to higher abstraction levels to implement concurrency

1) Implementing the **Runnable** or **Callable** interfaces & extending the **Thread** class.

2) Executor services with **completable future** objects.

3) Fork and Join framework.

4) Actor models like Akka.

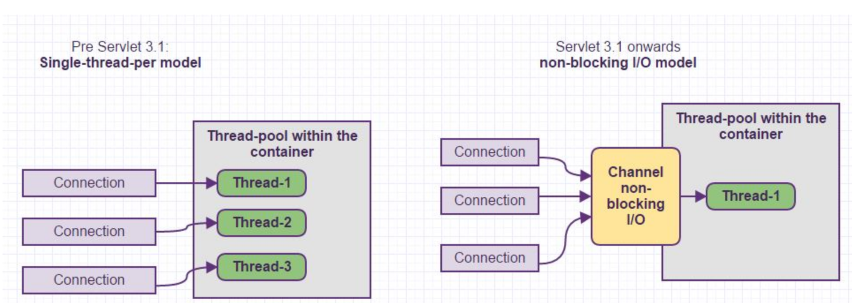
Q1. Can you give some scenarios where you had used multi-threading in Java applications?
A1.

Scenario 1: Servlets are inherently multi-threaded

and each user will be using a thread from the thread pool. The number of threads are configured via the web container of the application server. Servlet 3.1 supports non-blocking I/O for better throughput.

What is wrong with the thread-per-request model?

Pre Servlet 3.1 uses thread-per-request model, which limits the number of concurrent connections to the number of concurrently running JVM threads. Every thread introduces significant increase of memory footprint and CPU utilization via context switches. Servlet 3.1 rectifies this via non-blocking I/O. Fewer threads can be used in a pool to execute the request. NIO allows you to manage multiple channels (network connections or files) using only a single (or fewer) threads.

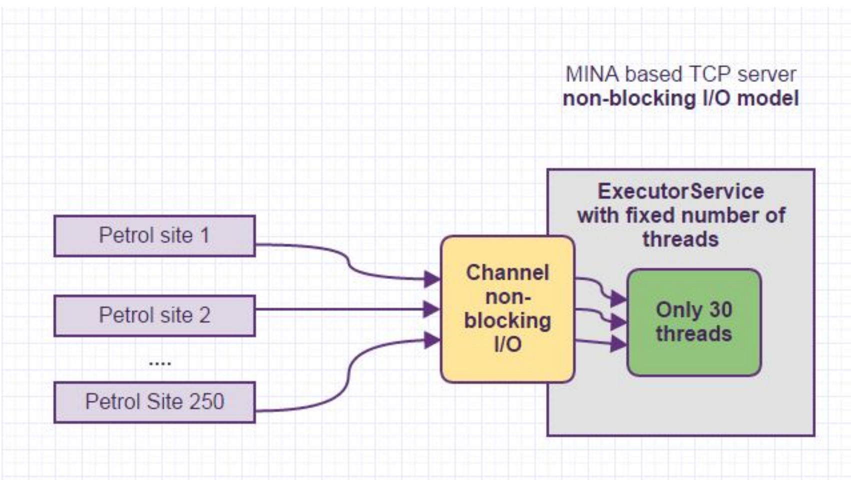


As a Java Architect

Java architecture & design concepts

Scenario 2: A MINA (i.e. a non-blocking I/O based) server with low level TCP protocol

to service 250+ petrol sites. The server supports the pay at the pump solution. The clients are C++ based and send fuel and credit card details to the server. A pool of reusable threads say 30 can be used to handle concurrent transactions.



Scenario 3: A Swing programmer deals with the following kinds of threads:

a) Initial threads that execute the initial application code.

b) The event dispatch thread, where all event-handling code is executed. Most code that interacts with the Swing framework must also execute on this thread.

c) Worker threads, also known as background threads, where time-consuming background tasks are executed. For example, loading an image, retrieving and caching the data, processing any time consuming logic, etc.

interview Q&As with diagrams | What should be a typical Java EE architecture?

Senior Java developers must have a good handle on

open all | close all
⊞ Best Practice (6)
⊞ Coding (26)
⊞ Concurrency (6)
⊞ Design Concepts (7)
⊞ Design Patterns (11)
⊞ Exception Handling (3
⊞ Java Debugging (21)
⊞ Judging Experience I
⊞ 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

Scenario 4: Asynchronous processing by spawning a worker thread

An online application with a requirement to produce time consuming reports or a business process (e.g. rebalancing accounts, aggregating hierachical information, etc) could benefit from making these long running operations asynchronous. These tasks are performed on a separate worker thread. Once the reports or the long running business process is completed, the outcome can be communicated to the user via emails or asynchronously refreshing the web page via techniques known as “server push” or “client pull”. A typical example would be

a) A user makes a request for an aggregate report or a business process like rebalancing his/her portfolios.

b) The user input can be saved to a database table for a separate process to periodically pick it up and process it asynchronously.

c) The user could now continue to perform other functionality of the website without being blocked.

d) A separate process running on the same machine or different machine can periodically scan the table for any entries and produce the necessary reports or execute the relevant business process. This could be a scheduled job that runs once during off-peak or every 10 minutes. This depends on the business requirement.

e) Once the report or the process is completed, notify the user via emails or making the report available online to be downloaded.

A **CountDownLatch** can be used to wait on multiple threads performig different tasks. Once CountDownLatch reaches zero, the waiting threads can be released. For example 3

JEE Tutorials (19)
Scala Tutorials (1)
Spring & HIbernate Tu
Tools Tutorials (19)
Other Tutorials (45)

100+ Preparing for pre-interview Java written home assignments & 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
Writing Code Home A
Written Test Core Jav
Written Test JEE (1)

How good are your…to go places?

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

separate threads populating the header, body, and footer sections. The CountDownLatch starts from 3.

Asynchronous (i.e. non-blocking) processing in Java examples

1. Asynchronous processing in Java real life examples – part-1

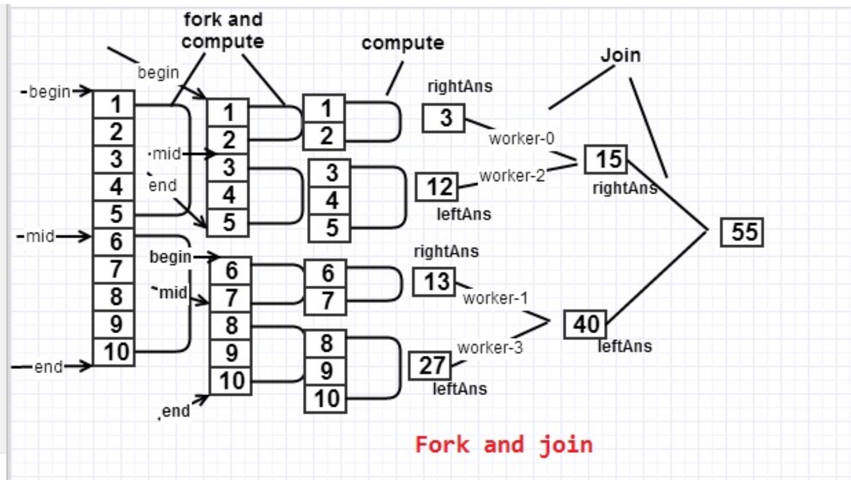
2. Asynchronous processing in Java real life examples – part-2

Scenario 5: Writing your own editor in Java where syntax highlighting is performed on a separate thread

To maximize the performance of the application, the CPU intensive syntax highlighting can be carried out on a separate worker thread whilst the user can use the editor.

Scenario 6: Java 7 fork and join to process computation intensive algorithms on a multi-core machine

Example. numbers = {1,2,3,4,5,6,7,8,9,10}, sum = 55; process them using the fork and join feature introduced in Java 7.



Pseudo Code for Fork, Compute and Join

```
Result solve(Problem problem) {
    if (problem is small) //e.g. smaller than ba
        compute the problem
    else {
        split problem into chunks
        fork new subtasks to solve each chunk
        join all subtasks
        compose result from subresults
    }
}
```

Here is the Java code

```

package com.fork.join;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.RecursiveTask;

class SumTask extends RecursiveTask<Integer> {

 static final int CHUNK_SIZE = 3; // execution b

 Integer[] numbers;
 int begin;
 int end;

 SumTask(Integer[] numbers, int begin, int end)
  this.numbers = numbers;
  this.begin = begin;
  this.end = end;
 }

 @Override
 protected Integer compute() {
```

```
  //sums the given number
  if (end - begin <= CHUNK_SIZE) {
    int sum = 0;
    List<Integer> processedNumbers = new ArrayLi
    for(int i=begin; i < end; ++i) {
      processedNumbers.add(numbers[i]);//just to
      sum += numbers[i];
    }

    //tracking thread, numbers processed, and su
    System.out.println(Thread.currentThread().ge
      Arrays.asList(processedNumbers) +  ", sum 
    return sum;
  }

  //create chunks, fork and join
  else {
    int mid = begin + (end - begin) / 2; //mid p
    SumTask left  = new SumTask(numbers, begin, 
    SumTask right = new SumTask(numbers, mid, en
    left.fork();           //asynchronously exe
    int leftAns = right.compute();
    int rightAns = left.join();  //returns the a
    System.out.println("leftAns=" + leftAns + " 
    return leftAns + rightAns;
  }

 }
}
```

Here is the test class with the main method

```
package com.fork.join;

import java.util.concurrent.ForkJoinPool;

public class SumTaskTest {

 public static void main(String[] args) {
   int numberOfCpuCores = Runtime.getRuntime().a
   ForkJoinPool forkJoinPool = new ForkJoinPool(

   Integer[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8,
   int sum = forkJoinPool.invoke(new SumTask(num

   System.out.println(sum);
 }

}
```

The **output**

```
ForkJoinPool-1-worker-1 proceeing [[6, 7]], sum 
ForkJoinPool-1-worker-3 proceeing [[8, 9, 10]], 
leftAns=27 + rightAns=13
ForkJoinPool-1-worker-2 proceeing [[3, 4, 5]], s
ForkJoinPool-1-worker-0 proceeing [[1, 2]], sum 
```

```
leftAns=12 + rightAns=3
leftAns=40 + rightAns=15
55

```

Q. Where to use fork/join as opposed to using the ***ExecutorService*** framework?

A. The Fork/Join Framework in Java 7 is designed for work that can be broken down into smaller tasks and the results of those tasks combined to produce the final result. Multicore processors are now widespread across server, desktop, and laptop hardware. They are also making their way into smaller devices, such as smartphones and tablets. Fork/Join offers serious gains for solving problems that involve **recursion**.

The fork/join tasks should operate as "pure" in-memory algorithms in which no I/O operations come into play. Also, communication between tasks through shared state should be avoided as much as possible, because that implies that locking might have to be performed. Ideally, tasks communicate only when one task forks another or when one task joins another.

ExecutorService continues to be a fine solution for many concurrent programming tasks, and in programming scenarios in which recursion is vital to processing power, it makes sense to use Fork/join. This fork and join feature is used in Java 8 parallel stream processing with lambda expressions.

Scenario 7: RESTful service processing the request asynchronously by spawning a new thread

```
@Controller
@RequestMapping(value = "/v1/myapp/processor", p
public class MyAppEndpointControllerImpl impleme

    @Inject
    @Named("replayExecutor")
    private Executor replayExecutor;
```

```java

    @Inject
    private MyAppReplayService replayService;

    @Override
    @ResponseStatus(HttpStatus.ACCEPTED)
    @RequestMapping(value = "/replay", method =
    public void replayPendingRequests(
        @RequestParam(required = false) final In

        replayExecutor.execute(new Runnable() {
            @Override
            public void run() {
                LOG.info("replayPendingRequests(
                replayService.replayPendingReque
            }
        });
    }

}


```

Configure the thread executor service via Spring JavaConfig.

```java
@Configuration
@EnableWebMvc
@Import({
    MyAppServiceConfiguration.class
})
@ComponentScan("com.mgl.wrap.ecm.enabler.endpoin
public class MyAppEndpointConfiguration extends

    @Bean
    public ExecutorService replayExecutor() {
        return Executors.newSingleThreadExecutor
    }

    //...
}


```

4 More scenario based

[Questions and Answers on Java multi-threading]()

Q2. Can you give some scenarios where you used the synchronized keyword in Java?

Q3. Is a multi-threading programming all bout speed?

Q4. How will you handle the following production issues?

Q5. Can you describe the multi-threading issues due to to deadlocks, thread starvation, and thread contention?

45+ Must know Java multi threading Interview Questions & Answers:

1. 15 Java multi-threading interview Q&A for beginner to intermediate level

2. 6 popular Java multi-threading interview Q&A with diagrams and code

3. 5 Basic multi-threading interview Q&A

4. Atomicity, Visibility, and Ordering in Java multi threading

5. threadLocal Interview questions & answers

6. How a thread gets blocked or suspended in Java?

7. Differences between X and Y interview Q&A on multi-threading

298 views

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

286 views

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

265 views

8 Git Source control system interview questions & answers

221 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.

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.

‹ 07: ♥ 20+ Pre interview refresher on productivity & debugging tools for Java developers

♦ Sample Java Resume or CV for Java Developers ›

Posted in FAQ Core Java Job Interview Q&A Essentials, Multithreading

2 comments on "02: ♥♦ 10+ Java multi-threading scenarios interview Q&As"

Amish says:

June 15, 2016 at 1:24 pm

I would like to enroll for 1 month subscription. Are you running any offer at the moement?

Reply

Arulkumaran Kumaraswamipillai says:

June 15, 2016 at 2:11 pm

Join my LinkedIn group (http://www.linkedin.com/groups/6978083) to get notified of any promotions & discount coupons. Also, I do promotions via my Facebook page.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

E-mail *

Website

Post Comment

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